RLDM 2017

3rd Multidisciplinary Conference on
Reinforcement Learning and Decision Making

June 11-14, 2017
The University of Michigan
Ann Arbor, Michigan, USA

$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$

EXTENDED ABSTRACTS

WWW.RLDM.ORG

TABLE OF CONTENTS

M1: Effective Warm Start for the Online Actor-Critic Reinforcement Learning based mHealth Intervention 6
M9: Efficient Reinforcement Learning via Initial Pure Exploration 11
M15: Reinforcement Learning in Rich-Observation MDPs using Spectral Methods 16
M19: Communications that Emerge through Reinforcement Learning Using a (Recurrent) Neural Network 21
M20: Functions that Emerge through End-to-end Reinforcement Learning — The Direction for Artificial General Intelligence — 26
M21: Unsupervised Basis Function Adaptation for Reinforcement Learning 31
M23: Deciding to Specialize and Respecialize a Value Function for Relational Reinforcement Learning 36
M25: Exploring by Believing 41
M27: Discovering Symmetries for Sample Efficient Reinforcement Learning 47
M29: Learning against sequential opponents in repeated stochastic games 52
M37: Artificial Improvisation: Improvisational Theatre with Deep Neural Networks and Robots 57
M41: Shared Learning in Ensemble Deep Q-Networks 62
M42: Aging of the Exploring Mind: Older Adults Deviate more from Optimality in Complex Choice Environments 67
M43: Efficient Parallel Methods for Deep Reinforcement Learning 72
M44: Combining Neural Networks and Tree Search for Task and Motion Planning in Challenging Environments 77
M55: Sufficient Markov Decision Processes with Alternating Deep Neural Networks 82
M58: Deep and Shallow Approximate Dynamic Programming 87
M59: Reinforcement Learning Algorithm for Patients with Type-1 Diabetes 92
M60: Fast Adaptation of Behavior to Changing Goals with a Gamma Ensemble 97
M61: Unlocking the Potential of Simulators: Design with RL in Mind 102

M62: Thompson Sampling for User-Guided Multi-Objective Bandits Optimization 107
M65: Spatial Sampling Strategies with Multiple Scientific Frames of Reference 112
M67: Neural Network Memory Architectures for Autonomous Robot Navigation 117
M70: Visualizing High-Dimensional MDPs with Model-Free Monte Carlo 122
M71: Mellowmax: An Alternative Softmax Operator for Reinforcement Learning 127
M72: Mechanisms of Overharvesting in Patch Foraging 132
M82: Stochastic Primal-Dual Methods and Sample Complexity of Markov Decision Processes 137
M83: Manipulating Model-based and Model-free Reinforcement Learning in Humans 143
M86: Policy Iteration for Discounted Reinforcement Learning Problems in Continuous Time and Space 148
M88: Automatically Deriving Rational Heuristics for Risky Choice 153
M93: SAIL: A Temporal Difference Approach to State Aware Imitation Learning 158
M94: A confidence-based reinforcement learning model for perceptual learning 163
M99: What is the nature of decision noise in random exploration? 168
M103: Effective, Time-Efficient State Representations for Human-Machine Decision Making 173

T3: New Reinforcement Learning Using a Chaotic Neural Network for Emergence of “Thinking” — “Exploration” Grows into “Thinking” through Learning 178
T7: Enhancing metacognitive reinforcement learning using reward structures and feedback 183
T8: Speeding Up HAM Learning with Internal Transitions 188
T16: Deep Reinforcement Learning with Model Learning and Monte Carlo Tree Search in Minecraft 194
T17: Using Advice in Model-Based Reinforcement Learning 199
T19: Bridging Computational Neuroscience and Machine Learning on Non-Stationary Multi-Armed Bandits 204
T20: Prediction under Uncertainty in Sparse Spectrum Gaussian Processes with Applications to Filtering and Control 209
T23: The neural mechanisms of worse than expected prediction errors 214
T27: Strategies of Observational Reinforcement Learning 219
T32: Shaping Model-Free Reinforcement Learning with Model-Based Pseudorewards 224
T36: Cross-Domain Transfer Learning using Target Apprentice 229
T39: Generalized Inverse Reinforcement Learning 234
T40: Adversarial Attacks on Deep Reinforcement Learning 239
T42: Learning Forest Wildfire Dynamics from Satellite Images Using Reinforcement Learning 244
T45: Goal-directed planning in a two-player game 249
T46: Assessing the Potential of Computational Modeling in Clinical Science 254
T50: Scalability of Deep Reinforcement Learning for Cooperative Control 259
T52: Prediction Regions and Tolerance Regions for Multi-Objective Markov Decision Processes 264
T55: Engagement matters: pupil and mental effort mediate depletion effect on subsequent physical tasks 269
T62: Attend, Adapt and Transfer: Attentive Deep Architecture for Adaptive Transfer from multiple sources in the same domain 274
T64: Decision-Making with Non-Markovian Rewards: From LTL to automata-based reward shaping 279
T65: Sample-Efficient Reinforcement Learning for Robot to Human Handover Tasks 284
T66: Robust Extensions to Policy Gradient Methods 289
T69: Hierarchical State Abstraction Synthesis for Discrete Models of Continuous Domains 294

T70: Learn to Survive by Deep Reinforcement Learning 299
T71: Towards Stability in Learning-based Control: A Bayesian Optimization-based Adaptive Controller 304
T76: Optimal Continuous State Planning with Semantic Observations 309
T81: Importance Sampling for Fair Policy Selection 314
T82: Multi-attribute decision making is best characterized by attribute-wise reinforcement learning model 319
T85: State Space Decomposition and Subgoal Creation for Transfer in Deep Reinforcement Learning 325
T88: A reward shaping method for promoting metacognitive learning 330
T89: Mouselab-MDP: A new paradigm for tracing how people plan 335
T92: Learning to (mis)allocate control: maltransfer can lead to self-control failure 340
T93: Helping people choose subgoals with sparse pseudo rewards 345
T96: Adversarially Robust Policy Learning through Active Construction of Physically-Plausible Perturbations 350

Effective Warm Start for the Online Actor-Critic Reinforcement Learning based mHealth Intervention

Feiyun Zhu
Department of Statistics,
Univeristy of Michigan, Ann Arbor, MI 48109.
Department of Computer Science & Engineering,
University of Texas at Arlington, TX 76013.
`fyzhu0915@gmail.com`

Peng Liao
Department of Statistics
Univeristy of Michigan
Ann Arbor, MI 48109
`pengliao@umich.edu`

Abstract

Online reinforcement learning (RL) is increasingly popular for the personalized mobile health (mHealth) intervention. It is able to personalize the type and dose of interventions according to user's ongoing statuses and changing needs. However, at the beginning of online learning, there are usually too few samples to support the RL updating, which leads to poor performances. A delay in good performance of the online learning algorithms can be especially detrimental in the mHealth, where users tend to quickly disengage with the mHealth app. To address this problem, we propose a new online RL methodology that focuses on an effective warm start. The main idea is to make full use of the data accumulated and the decision rule achieved in a former study. As a result, we can greatly enrich the data size at the beginning of online learning in our method. Such case accelerates the online learning process for new users to achieve good performances not only at the beginning of online learning but also through the whole online learning process. Besides, we use the decision rules achieved in a previous study to initialize the parameter in our online RL model for new users. It provides a good initialization for the proposed online RL algorithm. Experiment results show that promising improvements have been achieved by our method compared with the state-of-the-art method.

Keywords: Mobile Health (mHealth), Online learning, Reinforcement Learning (RL), Warm Start, Actor-Critic

Acknowledgements

The authors would like to thank the editor and the reviewers for their valuable suggestions. Besides, this work is supported by R01 AA023187, P50 DA039838, U54EB020404, R01 HL125440.

1 Introduction

With billions of smart device (i.e., smart-phones and wearable devices) users worldwide, mobile health (mHealth) interventions (MHI) are increasingly popular among the behavioral health, clinical, computer science and statistic communities [1, 2, 3, 4]. The MHI aims to make full use of smart technologies to collect, transport and analyze the raw data (weather, location, social activity, stress, urges to smoke, etc.) to deliver effective treatments that target behavior regularization [2]. For example, the goal of MHI is to optimally prevent unhealthy behaviors, such as alcohol abuse and eating disorders, and to promote healthy behaviors. Particularly, JITAIs (i.e., Just in time adaptive intervention) is especially interesting and practical due to the appealing properties [1]: (1) JITAIs could make adaptive and efficacious interventions according to user's ongoing statuses and changing needs; (2) JITAIs allow for the real-time delivery of interventions, which is very portable, affordable and flexible [5]. Therefore, JITAIs are widely used in a wide range of mHealth applications, such as physical activity, eating disorders, alcohol use, mental illness, obesity/weight management and other chronic disorders etc., that aims to guide people to lead healthy lives [4, 2, 6, 3, 7].

Normally, JITAIs is formed as an online sequential decision making (SDM) problem that is aimed to construct the optimal decision rules to decide when, where and how to deliver effective treatments [4, 2, 5]. This is a brand-new topic that lacks of methodological guidance. In 2014, Lei [1] made a first attempt to formulate the mHealth intervention as an online actor-critic contextual bandit problem. Lei's method is well suited for the small data set problem in the early stage of the mHealth study. However, this method ignores the important delayed effects of the SDM—the current action may affect not only the immediate reward but also the next states and, through that, all subsequent rewards [8]. To consider the delayed effects, it is reasonable to employ the reinforcement learning (RL) in the discount reward setting. RL is much more complex than the contextual bandit. It requires much more data to acquire good and stable decision rules [5]. However at the beginning of the online learning, there are too few data to start effective online learning. A simple and widely used method is to collect a fixed length of trajectory ($T_0 = 10$, say) via the micro-randomized trials [4], accumulating a few of samples, then starting the online updating. Such procedure is called the random warm start, i.e. RWS. However, there are two main drawbacks of the RWS: (1) it highly puts off the online RL learning before achieving good decision rules; (2) it is likely to frustrate the users because the random decision rules achieved at the beginning of online learning are not personalized to the users' needs. Accordingly, it is easy for users to abandon the mHealth app.

To alleviate the above problems, we propose a new online RL methodology by emphasizing effective warm starts. It aims to promote the performance of the online learning at the early stage and, through that, the final decision rule. The motivation is to make full use of the data and the decision rules achieved in the previous study, which is similar to the current study (cf. Sec. 3). Specifically, we use the decision rules achieved previously to initialize the parameter of the online RL learning for new users. The data accumulated in the former study is also fully used. As a result, the data size is greatly enriched at the beginning of our online learning algorithm. When the online learning goes on, the data gained from new users will have more and more weights to increasingly dominate the objective function. Our decision rule is still personalized according to the new user. Extensive experiment results show the power of the proposed method.

2 Markov Decision Process (MDP) and Actor-Critic Reinforcement Learning

MDP: The dynamic system (i.e. the environment) that RL interacts with is generally modeled as a Markov Decision Process (MDP) [8]. An MDP is a tuple $\{\mathcal{S}, \mathcal{A}, P, R, \gamma\}$ [9, 10, 11], where $\mathcal{S}$ is (finite) state space and $\mathcal{A}$ is (finite) action space. The state transition probability $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, from state s to the next state s' when taking action a, is given by $P(s, a, s')$. Let S_t, A_t and R_{t+1} be the random variables at time t representing the state, action and immediate reward respectively. The expected immediate reward $R(s, a) = \mathbb{E}(R_{t+1} \mid S_t = s, A_t = a)$ is assumed to be bounded over the state and action spaces [2]. $\gamma \in [0, 1)$ is a discount factor to reduce the influence of future rewards.

The stochastic policy $\pi(\cdot \mid s)$ decides the action to take in a given state s. The goal of RL is to interact with the environment to learn the optimal policy π^* that maximizes the total accumulated reward. Usually, RL uses the value function $Q^\pi(s, a) \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$ to quantify the quality of a policy π, which is the expected discounted cumulative reward, starting from state s, first choosing action a and then following the policy π: $Q^\pi(s, a) = \mathbb{E}\{\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a, \pi\}$. The value $Q^\pi(s, a)$ satisfies the following linear Bellman equation

$$Q^\pi(s, a) = \mathbb{E}_{s',a'|s,a,\pi}\{R(s, a) + \gamma Q^\pi(s', a')\} \tag{1}$$

The parameterized functions are generally employed to approximate the value and policy functions since [9] the system usually have too many states and actions to achieve an accurate estimation of value and policy. Instead they have to be iteratively estimated. Due to the great properties of quick convergences [10], the actor-critic RL algorithms are widely accepted to esimate the parameterized value $Q_{\mathbf{w}}(s, a) = \mathbf{w}^T \mathbf{x}(s, a) \approx Q^\pi$ and stochastic policy $\pi_\theta(\cdot \mid s) \approx \pi^*(\cdot \mid s)$, where $\mathbf{x}(s, a)$ is a feature function for the Q-value that merges the information in state s and action a. To learn the unknown parameters $\{\mathbf{w}, \theta\}$, we need a 2-step alternating updating rule until convergence: (1) the critic updating (i.e., policy evaluation) for $\mathbf{w}$ to estimate the Q-value function for the current policy, (2) the actor updating (i.e., policy improvement) for θ to search a better policy based on the newly estimated Q-value [10, 4].

Supposing the online learning for a new user is at decision point t, resulting in t tuples drawn from the MDP system, i.e., $\mathcal{D} = \{(s_i, a_i, r_i, s_i') \mid i = 1, \cdots, t\}$. Each tuple consists of four elements: the current state, action, reward and the next state. By using the data in $\mathcal{D}$, the Least-Squares Temporal Difference for Q-value (LSTDQ) [11, 8] is used for the *critic updating* to estimate $\widehat{\mathbf{w}}_t$ at time point t:

$$\widehat{\mathbf{w}}_t = \left[\zeta_c \mathbf{I} + \frac{1}{t}\sum_{i=1}^{t} \mathbf{x}_i \left(\mathbf{x}_i - \gamma \mathbf{y}_{i+1}\right)^{\intercal}\right]^{-1} \left(\frac{1}{t}\sum_{i=1}^{t} \mathbf{x}_i r_i\right), \tag{2}$$

where $\mathbf{x}_i = \mathbf{x}(s_i, a_i)$ is the feature at decision point i for the value function; $\mathbf{y}_{i+1} = \sum_{a \in \mathcal{A}} \mathbf{x}(s_{i+1}, a)\, \pi_{\widehat{\theta}_t}(a \mid s_{i+1})$ is the feature at the next time point; r_i is the immediate reward at the ith time point. By maximizing the average reward, i.e., a widely accepted criterion [10], we have the objective function for the *actor updating* (i.e., policy improvement)

$$\widehat{\theta}_t = \arg\max_{\theta} \frac{1}{t}\sum_{i=1}^{t}\sum_{a \in \mathcal{A}} Q(s_i, a; \widehat{\mathbf{w}}_t)\, \pi_\theta(a|s_i) - \frac{\zeta_a}{2}\|\theta\|_2^2 \tag{3}$$

where $Q(s_i, a; \widehat{\mathbf{w}}_t) = \mathbf{x}(s_i, a)^{\intercal}\, \widehat{\mathbf{w}}_t$ is the newly estimated value; ζ_c and ζ_a are the balancing parameters for the ℓ_2 constraint to avoid singular failures for the critic and actor update respectively. Note that after each actor update, the feature at the next time point $\mathbf{y}_{i+1}$ has to be re-calculated based on the newly estimated policy parameter $\widehat{\theta}_t$. When the discount factor $\gamma = 0$, the RL algorithm in (4), (5) is equivalent to the state-of-the-art contextual bandit method in the mHealth [1].

3 Our method

The actor-critic RL algorithm in (4), (5) works well when the sample size (i.e. t) is large. However at the beginning of the online learning, e.g., $t = 1$, there is only one tuple. It is impossible to do the actor-critic updating with so few samples. A popular and widely accepted method is to accumulate a small number of tuples via the micro-randomized trials [4] (called RWS). RWS is to draw a fixed length of trajectory ($T_0 = 10$, say) by applying the random policy with probability 0.5 to provide an intervention (i.e., $\mu(1 \mid s) = 0.5$ for all states s). RWS works to some extent, they are far from the optimal. One direct drawback with RWS is that it is very expensive in time to wait the micro-randomized trials to collect data from human, implying that we may still have a small number of samples to start the actor-critic updating. This tough problem badly affects the actor-critic updating not only at the beginning of online learning, but also along the whole learning process. Such case is due to the actor-critic objective functions is non-convex; any bad solution at the early online learning would bias the optimization direction, which easily leads some sub-optimal solution. Besides, the random policy in micro-randomized trials and the decision rules achieved at the early online learning is of bad user experience. Such problem makes it possible for the users to be inactive with or even to abandon the mHealth intervention.

To deal with the above problems, we propose a new online actor-critic RL methodology. It emphasizes effective warm starts for the online learning algorithm. The goal is to promote decision rules achieved at the early online learning stage and, through that, guide the optimization in a better direction, leading to a good final policy that is well suited for the new user. Specifically, we make full use of the data accumulated and decision rules learned in the previous study. Note that for the mHealth intervention design, there are usually several rounds of study; each round is pushed forward and slightly different from the former one. By using the data and policy gained in the former study, the RL learning in current study could quickly achieve good decision rules for new users, reducing the total study time and increasing the user experience at the beginning of the online learning.

Supposing that the former mHealth study is carried out in an off-policy, batch learning setting, we have $\bar{N}$ (40, say) individuals. Each individual is with a trajectory including $\bar{T} = 42$ tuples of states, actions and rewards. Thus in total there are $NT = \bar{N} \times \bar{T}$ tuples, i.e., $\bar{\mathcal{D}} = \{(\bar{s}_i, \bar{a}_i, \bar{r}_i, \bar{s}_i') \mid i = 1, \cdots, NT\}$. Besides the data in $\bar{\mathcal{D}}$, we employ the decision rule achieved in the former study to initialize the parameters in the current online learning. Note that we add a bar above the notations to distinguish the data obtained in the previous study from that of the current study.

At the tth decision point, we have both the data $\bar{\mathcal{D}}$ collected in the former study and the t new tuples drawn from the new user in $\mathcal{D}$ to update the online actor-critic learning. It has two parts: (1) the *critic updating* for $\widehat{\mathbf{w}}_t$ via

$$\widehat{\mathbf{w}}_t = \left\{\zeta_c \mathbf{I} + \frac{1}{t+1}\left[\frac{1}{NT}\sum_{j=1}^{NT} \bar{\mathbf{x}}_j \left(\bar{\mathbf{x}}_j - \gamma \bar{\mathbf{y}}_{i+1}\right)^{\intercal} + \sum_{i=1}^{t} \mathbf{x}_i \left(\mathbf{x}_i - \gamma \mathbf{y}_{i+1}\right)^{\intercal}\right]\right\}^{-1} \left[\frac{1}{t+1}\left(\frac{1}{NT}\sum_{j=1}^{NT} \bar{\mathbf{x}}_j \bar{r}_j + \sum_{i=1}^{t} \mathbf{x}_i r_i\right)\right] \tag{4}$$

and (2) the *actor updating* via

$$\widehat{\theta}_t = \arg\max_{\theta} \frac{1}{t+1}\left\{\frac{1}{NT}\sum_{j=1}^{NT}\sum_{a \in \mathcal{A}} Q(\bar{s}_j, a; \widehat{\mathbf{w}}_t)\, \pi_\theta(a|\bar{s}_j) + \sum_{i=1}^{t}\sum_{a \in \mathcal{A}} Q(s_i, a; \widehat{\mathbf{w}}_t) \cdot \pi(a|s_i)\right\} - \frac{\zeta_a}{2}\|\theta\|_2^2, \tag{5}$$

where $\{\bar{\mathbf{x}}_j\}_{j=1}^{NT}$ is data in the previous study; $(\mathbf{x}_i)_{i=1}^{t}$ is the data that is collected from the new user; $\bar{\mathbf{x}}_i = \bar{\mathbf{x}}(s_i, a_i)$ is the feature vector at decision point i for the value function; $\bar{\mathbf{y}}_{i+1} = \sum_{a\in\mathcal{A}} \mathbf{x}(\bar{s}_{i+1}, a)\, \pi_{\hat{\theta}_t}(a \mid \bar{s}_{i+1})$ is the feature at the next time point; $\bar{r}_i$ is the immediate reward at the ith point; $Q(\bar{s}_i, a; \widehat{\mathbf{w}}_t) = \mathbf{x}(\bar{s}_i, a)^{\intercal}\, \widehat{\mathbf{w}}_t$ is the newly updated value.

In (4) and (5), the terms in the blue ink indicate the the previous data, which is with a normalized weight $\frac{1}{NT}$. In this setting, all the data obtained in the former study is treated as one sample for the current online learning. When current online learning goes on (i.e., t increases), the data collected from the new user gradually dominates the objective functions. Thus, we are still able to achieve personalized JITAIs that is successfully adapted to each new user.

4 Experiments

To verify the performance, we compare our method (i.e., NWS-RL) with the conventional RL method with the random warm start (RWS-RL) on the HeartSteps application [3]. The HeartSteps is a 42-day mHealth intervention that encourages users to increase the steps they take each day by providing positive interventions, such as suggesting taking a walk after sedentary behavior. The actions are binary including $\{0, 1\}$, where $a = 1$ means providing active treatments, e.g., sending an intervention to the user's smart device, while $a = 0$ means no treatment [2].

4.1 Simulated Experiments

In the experiments, we draw T tuples from each user, i.e., $\mathcal{D}_T = \{(S_0, A_0, R_0), (S_1, A_1, R_1), \cdots, (S_T, A_T, R_T)\}$, where the observation S_t is a column vector with p elements . The initial states and actions are generated by $S_0 \sim \mathrm{Normal}_p\{0, \Sigma\}$ and $A_0 = 0$, where $\Sigma = \begin{bmatrix} \Sigma_1 & 0 \\ 0 & I_{p-3} \end{bmatrix}$ and $\Sigma_1 = \begin{bmatrix} 1 & 0.3 & -0.3 \\ 0.3 & 1 & -0.3 \\ -0.3 & -0.3 & 1 \end{bmatrix}$. For $t \geq 1$, we have the state generation model and immediate reward model as follows

$$\begin{aligned} S_{t,1} &= \beta_1 S_{t-1,1} + \xi_{t,1}, \\ S_{t,2} &= \beta_2 S_{t-1,2} + \beta_3 A_{t-1} + \xi_{t,2}, \\ S_{t,3} &= \beta_4 S_{t-1,3} + \beta_5 S_{t-1,3} A_{t-1} + \beta_6 A_{t-1} + \xi_{t,3}, \\ S_{t,j} &= \beta_7 S_{t-1,j} + \xi_{t,j}, \quad \text{for } j = 4, \ldots, p \end{aligned} \tag{6}$$

$$R_t = \beta_{14} \times [\beta_8 + A_t \times (\beta_9 + \beta_{10} S_{t,1} + \beta_{11} S_{t,2}) + \beta_{12} S_{t,1} - \beta_{13} S_{t,3} + \varrho_t], \tag{7}$$

where $-\beta_{13} O_{t,3}$ is the treatment fatigue [4, 2]; $\{\xi_{t,i}\}_{i=1}^{p} \sim \mathrm{Normal}\left(0, \sigma_s^2\right)$ at the tth point is the noise in the state transition (6) and $\varrho_t \sim \mathrm{Normal}\left(0, \sigma_r^2\right)$ is the noise in the immediate reward model (7). To generate N different users, we need N different MDPs specified by the value of $\boldsymbol{\beta}$s in (6) and (7). The $\boldsymbol{\beta}s$ are generated in the following two steps: (a) set a basic $\boldsymbol{\beta}_{\text{basic}} = [0.40, 0.25, 0.35, 0.65, 0.10, 0.50, 0.22, 2.00, 0.15, 0.20, 0.32, 0.10, 0.45, 1.50, 800]$; (b) to obtain N different $\boldsymbol{\beta}$s (i.e., users or MDPs), we set the $\{\boldsymbol{\beta}_i\}_{i=1}^{N}$ as $\boldsymbol{\beta}_i = \boldsymbol{\beta}_{\text{basic}} + \boldsymbol{\delta}_i$, for $i \in \{1, 2, \cdots, N\}$, where $\boldsymbol{\delta}_i \sim \mathrm{Normal}(0, \sigma_b \mathbf{I}_{14})$, σ_b controls how different the users are and $\mathbf{I}_{14}$ is an identity matrix with 14×14 elements. To generate the MDP for a future user, we will also use this kind of method to generate new $\boldsymbol{\beta}$s.

4.2 Experiment Settings

The expectation of long run average reward (ElrAR) $\mathbb{E}[\eta^{\pi_{\hat{\theta}}}]$ is used to evaluate the quality of an estimated policy $\pi_{\widehat{\theta}}$ on a set of N=50 individuals. Intuitively, the ElrAR measures how much average reward in the long run we could totally get by using the learned policy $\pi_{\widehat{\theta}}$ for a number of users. In the HeartSteps application, the ElrAR measures the average steps that users take each day in a long period of time; a larger ElrAR corresponds to a better performance. The average reward $\eta^{\pi_{\hat{\theta}}}$ is calculated by averaging the rewards over the last $4,000$ elements in a trajectory of $5,000$ tuples under the policy $\pi_{\widehat{\theta}}$. Then ElrAR $\mathbb{E}[\eta^{\pi_{\hat{\theta}}}]$ is achieved by averaging the 50 $\eta^{\pi_{\hat{\theta}}}$'s.

In the experiment, we assume the parameterized policy in the form $\pi_\theta(a \mid s) = \exp[a\theta^{\intercal}\phi(s)] / \{1 + \exp[\theta^{\intercal}\phi(s)]\}$, where $\theta \in \mathbb{R}^q$ is the unknown variance and $\phi(s) = [1, s^{\intercal}]^{\intercal} \in \mathbb{R}^q$ is the feature function for policies that stacks constant 1 with the state vector s. The number of individuals in the former study is $\bar{N} = 40$. Each is with a trajectory of $\bar{T}$=42 time points. For the current study (cf., Table 1), there are $N = 50$ individuals. RWS has to accumulate tuples till $T_0 = 5$ and 10 respectively to start the online learning. Our method (i.e., NWS) has the ability to start the RL online learning algorithm immediately when the 1st tuple is available. Since the comparison of early online learning is our focuses, we set the total trajectory length for the online learning as $T = 30$ and $T = 50$, respectively. The noises are set $\sigma_s = \sigma_r = 1$ and $\sigma_\beta = 0.005$. Other variances are $p = 3$, $q = 4$, $\zeta_a = \zeta_c = 10^{-5}$. The feature processing for the value estimation is $\mathbf{x}(s, a) = [1, s^{\intercal}, a, s^{\intercal}a]^{\intercal} \in \mathbb{R}^{2p+2}$ for all the compared methods. Table 1 summarizes the experiment results of three methods: RWS-RL$_{T_0=5}$, RWS-RL$_{T_0=10}$ and NWS-RL$_{T_0=1}$. It includes two sub-tables: the left one shows the results of early online learning results, i.e., $T = 30$ and the right displays the results when $T = 50$. As we shall see, the proposed warm

Table 1: The average reward of three online RL methods as discount factor γ rises from 0 to 0.95: (a) Random Warm Start RL (RWS-RL) when $T_0 = 5$ and $T_0 = 10$ respectively; (b) the proposed New Warm Start RL (NWS-RL) is able to start the online learning when the 1st tuple is available, i.e., $T_0 = 1$. The **Red value** is the best and the *blue value* is the 2nd best.

γ	Average reward when trajectory length $T = 30$			Average reward when trajectory length $T = 50$		
	RWS-RL$_{T_0=5}$	RWS-RL$_{T_0=10}$	NWS-RL$_{T_0=1}$	RWS-RL$_{T_0=5}$	RWS-RL$_{T_0=10}$	NWS-RL$_{T_0=1}$
0	1152.9±18.8	*1349.7±13.9*	**1367.6±8.30**	1152.6±19.8	*1335.8±8.50*	**1365.2±8.30**
0.2	1152.4±22.2	*1320.2±26.8*	**1339.3±13.6**	1153.8±19.9	*1325.4±14.2*	**1361.4±8.70**
0.4	1149.1±23.8	*1300.6±32.6*	**1337.7±23.9**	1149.7±17.5	*1308.0±19.7*	**1335.1±15.8**
0.6	1155.2±29.0	*1301.1±32.7*	**1405.4±49.1**	1160.2±21.3	*1281.2±27.3*	**1387.6±48.3**
0.8	1267.8±43.3	*1326.7±29.8*	**1481.9±31.6**	1263.0±37.1	*1333.3±43.1*	**1501.2±35.5**
0.95	1327.9±46.0	*1354.9±27.5*	**1426.7±30.6**	1320.8±47.2	*1427.7±33.7*	**1479.3±39.7**
Avg.	1200.9	*1325.5*	**1393.1**	1200.0	*1335.2*	**1405.0**

The value of γ specifies different RL methods: (a) $\gamma = 0$ means the contextual bandit [1], (b) $0 < \gamma < 1$ indicates the discounted reward RL.

start method (NWS-RL) has an obvious advantage over the conventional RWS-RL, averagely achieving an improvement of 67.57 steps for $T = 30$ and 69.72 steps for $T = 50$ compared with the 2nd best policy in *blue*.

5 Conclusion and *Discussion*

In this paper, we propose a new online actor-critic reinforcement learning methodology for the mHealth application. The main idea is to provide an effective warm start method for the online RL learning. The state-of-the-art RL method for mHealth has the problem of lacking samples to start the online learning. To solve this problem, we make full use of the data accumulated and decision rules achieved in the former study. As a result, the data size is greatly enriched even at the beginning of online learning. Our method is able to start the online updating when the first tuple is available. Experiment results verify that our method achieves clear gains compared with the state-of-the-art method. In the future, we may explore the robust learning [12, 13] and graph learning [14, 15] on the online actor-critic RL learning algorithm.

References

[1] H. Lei, A. Tewari, and S. Murphy, “An actor-critic contextual bandit algorithm for personalized interventions using mobile devices,” in *NIPS 2014 Workshop: Personalization: Methods and Applications*, pp. 1 – 9, 2014.

[2] S. A. Murphy, Y. Deng, E. B. Laber, H. R. Maei, R. S. Sutton, and K. Witkiewitz, “A batch, off-policy, actor-critic algorithm for optimizing the average reward,” *CoRR*, vol. abs/1607.05047, 2016.

[3] W. Dempsey, P. Liao, P. Klasnja, I. Nahum-Shani, and S. A. Murphy, “Randomised trials for the fitbit generation,” *Significance*, vol. 12, pp. 20 – 23, Dec 2016.

[4] P. Liao, A. Tewari, and S. Murphy, “Constructing just-in-time adaptive interventions,” *Phd Section Proposal*, pp. 1–49, 2015.

[5] F. Zhu, P. Liao, X. Zhu, Y. Yao, and J. Huang, “Cohesion-based online actor-critic reinforcement learning for mhealth intervention,” *arXiv:1703.10039*, 2017.

[6] H. Lei, *An Online Actor Critic Algorithm and a Statistical Decision Procedure for Personalizing Intervention*. PhD thesis, University of Michigan, 2016.

[7] D. Gustafson, F. McTavish, M. Chih, A. Atwood, R. Johnson, M. B. ..., and D. Shah, “A smartphone application to support recovery from alcoholism: a randomized clinical trial,” *JAMA Psychiatry*, vol. 71, no. 5, 2014.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2nd ed., 2012.

[9] M. Geist and O. Pietquin, “Algorithmic survey of parametric value function approximation,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 6, pp. 845–867, 2013.

[10] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, “A survey of actor-critic reinforcement learning: Standard and natural policy gradients,” *IEEE Trans. Systems, Man, and Cybernetics*, vol. 42, no. 6, pp. 1291–1307, 2012.

[11] M. G. Lagoudakis and R. Parr, “Least-squares policy iteration,” *J. of Machine Learning Research (JLMR)*, vol. 4, pp. 1107–1149, 2003.

[12] F. Zhu, B. Fan, X. Zhu, Y. Wang, S. Xiang, and C. Pan, “10,000+ times accelerated robust subset selection (ARSS),” in *Proc. Assoc. Adv. Artif. Intell. (AAAI)*, pp. 3217–3224, 2015.

[13] Y. Wang, C. Pan, S. Xiang, and F. Zhu, “Robust hyperspectral unmixing with correntropy-based metric,” *IEEE Transactions on Image Processing*, vol. 24, no. 11, pp. 4027–4040, 2015.

[14] F. Zhu, Y. Wang, S. Xiang, B. Fan, and C. Pan, “Structured sparse method for hyperspectral unmixing,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 88, pp. 101–118, 2014.

[15] H. Li, Y. Wang, S. Xiang, J. Duan, F. Zhu, and C. Pan, “A label propagation method using spatial-spectral consistency for hyperspectral image classification,” *International Journal of Remote Sensing*, vol. 37, no. 1, pp. 191–211, 2016.

Efficient Reinforcement Learning via Initial Pure Exploration

Sudeep Raja Putta
Conduent Labs India
Bangalore, India
`sudeepraja94@gmail.com`

Theja Tulabandhula
University of Illinois at Chicago
Chicago, IL 60607
`tt@theja.org`

Abstract

In several realistic situations, an interactive learning agent can practice and refine its strategy before going on to be evaluated. For instance, consider a student preparing for a series of tests. She would typically take a few practice tests to know which areas she needs to improve upon. Based of the scores she obtains in these practice tests, she would formulate a strategy for maximizing her scores in the actual tests. We treat this scenario in the context of an agent exploring a fixed-horizon episodic Markov Decision Process (MDP), where the agent can practice on the MDP for some number of episodes (not necessarily known in advance) before starting to incur regret for its actions.

During practice, the agent's goal must be to maximize the probability of following an optimal policy. This is akin to the problem of Pure Exploration (PE). We extend the PE problem of Multi Armed Bandits (MAB) to MDPs and propose a Bayesian algorithm called Posterior Sampling for Pure Exploration (PSPE), which is similar to its bandit counterpart. We show that the Bayesian simple regret converges at an optimal exponential rate when using PSPE.

When the agent starts being evaluated, its goal would be to minimize the cumulative regret incurred. This is akin to the problem of Reinforcement Learning (RL). The agent uses the Posterior Sampling for Reinforcement Learning algorithm (PSRL) initialized with the posteriors of the practice phase. We hypothesize that this PSPE + PSRL combination is an optimal strategy for minimizing regret in RL problems with an initial practice phase. We show empirical results which prove that having a lower simple regret at the end of the practice phase results in having lower cumulative regret during evaluation.

Keywords: Markov Decision Process, Multi Armed Bandit, Pure Exploration,
Reinforcement Learning

1 Introduction

In problems involving sequential decision making under uncertainty, there exist at least two different objectives: a) optimize the online performance, and b) find an optimal behavior. In the context of Multi Armed Bandits (MAB), these objectives correspond to: a) maximize cumulative reward, and b) identify the best arm. The first objective is the widely studied problem of cumulative regret minimization. The second one is a Pure Exploration (PE) problem, where agent has to efficiently gather information to identify the optimal arm. For Markov Decision Processes (MDPs), the first objective is the classical Reinforcement Learning (RL) problem. On the other hand, PE in MDPs is an area which has not been explored in detail.

Our first contribution is that we propose an algorithm called Posterior Sampling for Pure Exploration (PSPE), for PE in fixed-horizon episodic MDPs. We define an objective similar to the notion of simple regret in MABs and analyze its convergence when using PSPE. In PSPE, the agent's goal is to explore the MDP such that it maximizes the probability of following an optimal policy after some number of episodes (not necessarily known in advance). The following table captures PSPE's relation to other algorithms.

	RL	PE
MAB	TS [5]	PTS[4]
MDP	PSRL [2]	PSPE

Thompson Sampling (TS) [5] is a Bayesian algorithm for maximizing the cumulative rewards received in bandits. The idea is to pull an arm according to its confidence (i.e, its probability of being optimal the optimal arm). It maintains a prior distribution over bandit instances. At each step, it samples an instance of a bandit from the posterior and pulls its optimal arm. Pure exploration Thompson Sampling (PTS)[4] modifies TS by adding a re-sampling step. TS is not suitable for PE as it pulls the estimated best arm almost all the time. It takes a very long time to ascertain that none of the other arms offer better rewards. The re-sampling step prevents pulling the estimated best arm too often and helps in achieving a higher confidence in lesser number of arm pulls. Posterior Sampling for Reinforcement Learning (PSRL) [2] extends TS for the RL problem on episodic fixed-horizon MDPs. It maintains a prior distribution over MDPs. At the beginning of each episode, it samples a MDP instance from the posterior, finds its optimal policy using dynamic programming and acts according to this policy for the duration of the episode. It updates the posterior with the rewards and transitions witnessed. For PE in MDPs, we propose PSPE, which adds a re-sampling step to PSRL.

In reality however, agents may have a different objective: **Optimize online performance after a period of exploration without regret**. For instance, consider a student preparing for a series of tests. She would typically take a few practice tests to know which areas she needs to improve upon. Based of the scores she obtains in these practice tests, she would formulate a strategy for maximizing her scores in the actual tests. Another example is a robot in a robotics competition. It typically has a few practice rounds before the evaluation rounds. In the practice rounds, the robot can freely explore the environment such that it maximizes its score in the evaluation round.

For this new objective, we claim that the best strategy is to use PSPE during practice and switch to PSRL during evaluation. This is our second contribution. At the end of the practice phase, PSPE maximizes the probability of following an optimal policy. It essentially initializes the priors of PSRL such that they are very close to the true MDP. PSRL can thus leverage these priors to obtain near optimal rewards.

2 Episodic Fixed Horizon MDP

An episodic fixed horizon MDP M is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, R, P, H, \rho \rangle$. Here $\mathcal{S} = \{1, ..., S\}$ and $\mathcal{A} = \{1, ..., A\}$ are finite sets of states and actions respectively. The agent interacts with the MDP in episodes of length H. The initial state distribution is given by ρ. In each step $h = 1, ..., H$ of an episode, the agent observes a state s_h and performs an action a_h. It receives a reward r_h sampled from the reward distribution $R(s_h, a_h)$ and transitions to a new state s_{h+1} sampled from the transition probability distribution $P(s_h, a_h)$. The average reward received for a particular state-action is $\bar{R}(s,a) = \mathbb{E}[r | r \sim R(s,a)]$.

For fixed horizon MDPs, a policy π is a mapping from $\mathcal{S}$ and $\{1, ..., H\}$ to $\mathcal{A}$. The value of a state s and action a under a policy π is: $Q_\pi(s,a,h) = \mathbb{E}\left[\bar{R}(s,a) + \sum_{i=h+1}^{H} \bar{R}(s_i, \pi(s_i, i))\right]$. Let $V_\pi(s,h) = Q_\pi(s, \pi(s,h), h)$. A policy π^* is an optimal policy for the MDP if $\pi^* \in \arg\max_\pi V_\pi(s,h)$ for all $s \in \mathcal{S}$ and $h = 1, ..., H$. Let the set of optimal policies be Π^*. For a MDP M, let Π_M be the set of optimal policies.

3 Posterior Sampling for Reinforcement Learning

Consider a MDP with S states, A actions and horizon length H. PSRL maintains a prior distribution on the set of MDPs $\mathcal{M}$, i.e on the reward distribution R (on SA variables) and the transition probability distribution P (on S^2A variables). At the beginning of each episode t, a MDP M_t is sampled from the current posterior. Let P_t and R_t be the transition and reward distributions of M_t. The set of optimal policies Π_{M_t} for this MDP can be found using Dynamic Programming as P_t and R_t are known. The agent samples a policy π_t from Π_{M_t} and follows it for H steps. The rewards and transitions witnessed during this episode are used to update the posteriors. Let f be the prior density over the MDPs and $\mathcal{H}_t$ be the history of episodes seen until $t-1$. Let $s_{h,t}$ be the state observed, $a_{h,t}$ be the action performed and $r_{h,t}$ be the reward received at time h in episode t.

Like TS, PSRL maintains a prior distribution over the model, in this case a MDP. At each episode, it samples a model from the posterior and acts greedily according to the sample. TS selects arms according to their posterior probability of being optimal and PSRL selects policies according to the posterior probability they are optimal. It is possible to compute the posterior efficiently and sample from it by a proper choice of conjugate prior distributions or by the use of Markov Chain Monte Carlo methods.

Algorithm 1 PSRL

```
$\mathcal{H}_1 = \{\}$
for $t = 1, 2, ...$ do
    Sample $M_t \sim f(\cdot|\mathcal{H}_t)$
    Choose a policy $\pi_t$ at random from $\Pi_{M_t}$
    Observe initial state $s_{1,t}$
    for $h = 1, ..., H$ do
        Perform action $a_{h,t} = \pi_t(s_{h,t}, h)$
        Observe reward $r_{h,t}$ and next state $s_{h+1,t}$
    end for
    $\mathcal{H}_{t+1} = \mathcal{H}_t \cup \{(s_{h,t}, a_{h,t}, r_{h,t}, s_{h+1,t})|h = 1..H\}$
end for
```

Algorithm 2 PSPE

```
$\mathcal{H}_1 = \{\}$
for $t = 1, 2, ...$ do
    Sample $M_t \sim f(\cdot|\mathcal{H}_t)$
    Sample $B \sim Bernoulli(\beta)$
    if $B = 1$ then
        Choose a policy $\pi_t$ at random from $\Pi_{M_t}$
    else
        repeat
            Re-sample $\widetilde{M_t} \sim f(\cdot|\mathcal{H}_t)$
        until $\Pi_{\widetilde{M_t}} - \Pi_{M_t} \neq \emptyset$
        Choose a policy $\pi_t$ at random from $\Pi_{\widetilde{M_t}} - \Pi_{M_t}$
    end if
    Observe initial state $s_{1,t}$
    for $h = 1, ..., H$ do
        Perform action $a_{h,t} = \pi_t(s_{h,t}, h)$
        Observe reward $r_{h,t}$ and next state $s_{h+1,t}$
    end for
    $\mathcal{H}_{t+1} = \mathcal{H}_t \cup \{(s_{h,t}, a_{h,t}, r_{h,t}, s_{h+1,t})|h = 1..H\}$
end for
```

4 Posterior Sampling for Pure Exploration

PSRL is not suitable for PE as after a certain point, it almost certainly follows the optimal policy and does not spend much effort in refining its knowledge of other policies. PSPE modifies PSRL by adding a re-sampling step. This is an extension of the Top-Two sampling idea of PTS to PSRL. This prevents it from following an estimated optimal policy too frequently.

The algorithm depends on a parameter β, where $0 < \beta < 1$, which controls how often an optimal policy of the sampled MDP is followed. At each episode t, PSPE samples a MDP M_t and finds its set of optimal policies Π_{M_t}. With probability β it follows a policy from this set. With probability $1-\beta$ it re-samples MDPs until a different set of policies $\Pi_{\widetilde{M_t}}$ is obtained. It then follows a policy from the set $\Pi_{\widetilde{M_t}} - \Pi_{M_t}$ for H steps. In the case of bandits, PSPE is equivalent to PTS. PSRL is the same as PSPE with $\beta = 1$.

5 Analysis

Let $x_\pi(M) = |\Pi_M \cap \{\pi\}|/|\Pi_M|$. The confidence of policy π after episode t is: $\alpha(\pi,t) = \int_{M\in\mathcal{M}} x_\pi(M) f(M|\mathcal{H}_{t+1})dM$. The mean episodic reward of a policy π is: $\mu(\pi) = \sum_{s\in\mathcal{S}} \rho(s) V_\pi(s,1)$. Let $\mu^* = \max_\pi \mu(\pi)$. The gap of a policy is $\Delta(\pi) = \mu^* - \mu(\pi)$ for $\pi \notin \Pi^*$. The simple regret after the episode t is : $r_t = \mu^* - \sum_\pi \alpha(\pi,t)\mu(\pi)$. Let $\Theta(t)$ be the confidence of the sub-optimal policies: $\Theta(t) = \sum_{\pi\notin\Pi^*} \alpha(\pi,t)$. We re-write r_t as:

$$\begin{aligned} r_t &= \mu^* - \sum_\pi \alpha(\pi,t)\mu(\pi) = \mu^* - \sum_{\pi\notin\Pi^*} \alpha(\pi,t)\mu(\pi) - \sum_{\pi\in\Pi^*} \alpha(\pi,t)\mu^* = (1 - \sum_{\pi\in\Pi^*} \alpha(\pi,t))\mu^* - \sum_{\pi\notin\Pi^*} \alpha(\pi,t)\mu(\pi) \\ &= \Theta(t)\mu^* - \sum_{\pi\notin\Pi^*} \alpha(\pi,t)\mu(\pi) \end{aligned}$$

Upper and lower bounds for r_t can be expressed in terms of $\Theta(t)$ and $\Delta(\pi)$:

$$\Theta(t) \min_\pi \Delta(\pi) \leq \Theta(t)\mu^* - \sum_{\pi\notin\Pi^*} \alpha(\pi,t)\mu(\pi) \leq \Theta(t) \max_\pi \Delta(\pi) \implies \Theta(t) \min_\pi \Delta(\pi) \leq r_t \leq \Theta(t) \max_\pi \Delta(\pi)$$

r_t is bounded above and below by $\Theta(t)$ asymptotically. The convergence of $\Theta(t)$ dictates the convergence of r_t.

We use results from the analysis of PTS[4] about the convergence of $\Theta(t)$.

There exist constants $\{\Gamma^*_\beta > 0 : \beta \in (0,1)\}$ such that $\Gamma^* = \max_\beta \Gamma^*_\beta$ exists, $\beta^* = \arg\max_\beta \Gamma^*_\beta$ is unique for a given MDP and the following hold with probability 1:

1. Under PSPE with parameter β^*, $\lim\limits_{t\to\infty} -\frac{1}{t}\log\Theta(t) = \Gamma^*$. Under any algorithm, $\limsup\limits_{t\to\infty} -\frac{1}{t}\log\Theta(t) \le \Gamma^*$
2. Under PSPE with parameter $\beta \in (0,1)$, $\lim\limits_{t\to\infty} -\frac{1}{t}\log\Theta(t) = \Gamma^*_\beta$
3. $\Gamma^* \le 2\Gamma^*_{\frac{1}{2}}$ and $\dfrac{\Gamma^*}{\Gamma^*_\beta} \le \max\left\{\dfrac{\beta^*}{\beta}, \dfrac{1-\beta^*}{1-\beta}\right\}$

$\Theta(t)$ cannot converge faster than $\exp(-t\Gamma^*)$. When using PSPE with parameter β, $\Theta(t)$ converges at rate of Γ^*_β in the limit. When $\beta = \beta^*$, this rate of convergence is Γ^*, which is optimal. When β is close to β^*, Γ^*_β is close to Γ^*. In particular, the choice of $\beta = 1/2$ is robust as $\Gamma^*_{1/2}$ is atleast half of Γ^* for any MDP.

6 Experiments

We compare the performance of PSPE with different values of β and random exploration. To ease the procedure of computing posterior distributions and sampling MDPs from the posterior, we use suitable conjugate-prior distributions. For the transition probabilities, we use a uniform Dirichlet prior and a categorical likelihood, and for reward distribution, we use a Gaussian prior ($\mathcal{N}(0,1)$) and a Gaussian likelihood with unit variance. We calculate the simple regret by sampling 1000 independent MDPs from the posterior and approximating $\alpha(\pi, t)$ using sample means. All the results are averaged across 50 trials.

Stochastic Chains [Figure 1], are a family of MDPs which consist of a long chain of N states. At each step, the agent can choose to go left or right. The left actions (indicated by thick lines) are deterministic, but the right actions (indicated by dotted lines) result in going right with probability $1 - 1/N$ or going left with probability $1/N$. The only two rewards in this MDP are obtained by choosing left in state 1 and choosing right in state N. These rewards are drawn from a normal distribution with unit variance. Each episode is of length $H = N$. The agent begins each episode at state 1. The optimal policy is to go right at every step to receive an expected reward of $(1 - \frac{1}{N})^{N-1}$. For the RL problem on these MDPs, dithering strategies like ϵ-greedy or Boltzmann exploration are highly inefficient and could lead to regret that grows exponentially in chain length.

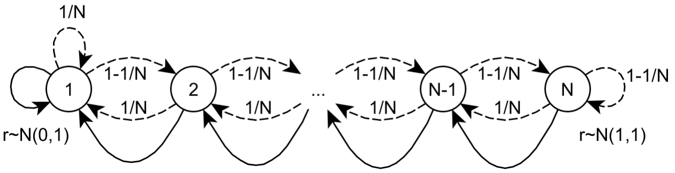


Figure 1: Stochastic Chain

We consider a stochastic chain of length 10. The total number of deterministic policies for this MDP are $2^{10\times 10}$. We plot the simple regret of PSPE with $\beta = [0.0, 0.25, 0.5, 0.75, 0.1]$ and random exploration for 1000 episodes in Figure 2. For this MDP, β^* appears to be close to 0.25 as the simple regret converges at the fastest rate when $\beta = 0.25$. As values of β closer to β^* have a faster rate of convergence, the convergence of $\beta = 0.0$ and $\beta = 0.5$ is similar. For PSRL, which is $\beta = 1.0$, the convergence is much slower. Random exploration however, is highly inefficient. This is because PSPE is able to achieve "Deep Exploration" [?] whereas random exploration does not. Deep Exploration means that the algorithm selects actions which are oriented towards positioning the agent to gain useful information further down in the episode.

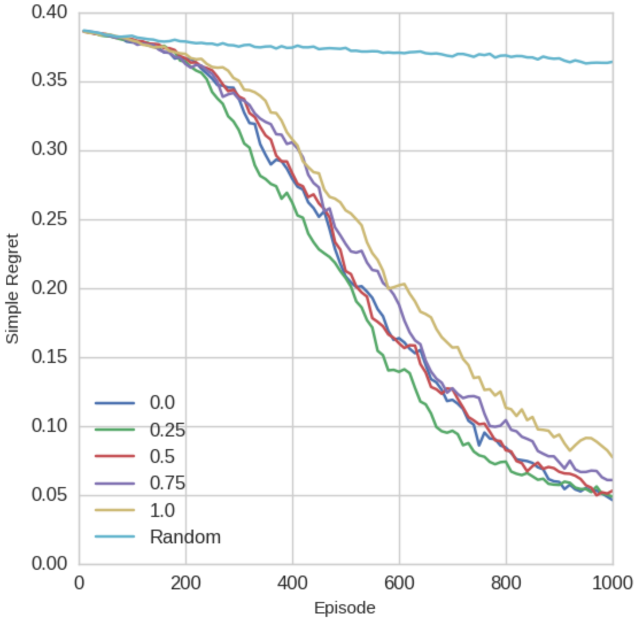


Figure 2: Simple Regret of Stochastic Chain

7 Reinforcement Learning with Practice

In this section, we try to answer the question: "When does it make sense for the agent to use PSPE?". Consider the following situation: The agent's goal is to maximize the cumulative reward, but the rewards are accumulated from the Tth episode. The rewards obtained during episodes 1 to $T-1$ are not used to evaluate the agent's performance.

The first $T-1$ episodes can be considered as practice, where the agent gathers information so that it obtains near optimal rewards from episode T. The agent may not know T in advance. It will be told at the beginning of the Tth episode that its performance is being evaluated. It is not entirely apparent which strategy the agent should use during practice. The agent could ignore the fact that the rewards accumulated during practice do not matter and always use a reward maximizing strategy such as PSRL. We argue that the best strategy is to use PSPE during practice and switching to PSRL during evaluation. Logically, having lower simple regret after practice should result in lower cumulative regret during evaluation. Since PSPE with parameter β^* reaches a lower simple regret faster than PSRL, an optimal policy of a sampled MDP will be an optimal policy of the true MDP with high probability. Hence, we claim that lower regret is incurred by PSRL in the evaluation when PSPE is used during practice.

Like before, we consider a stochastic chain of length 10. We let the practice phase last for different intervals starting from 0 to 1000 in steps of 10. During practice, the agents use PSPE with with $\beta = [0.0, 0.25, 0.5, 0.75, 0.1]$. After practice, the agents use PSRL for 1000 episodes. The cumulative regret of these agents after the 1000 episodes is plotted against the simple regret at the end of practice. The simple and cumulative regrets are highly correlated, as show in Figure 3.

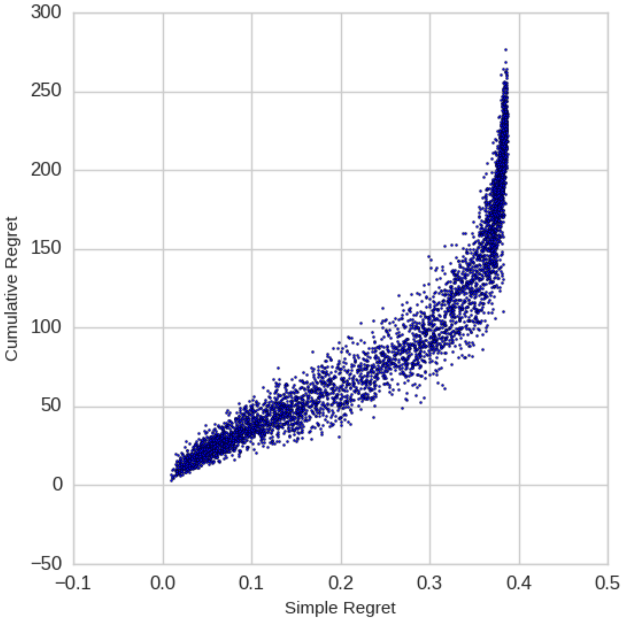


Figure 3: Simple Regret vs Cumulative Regret

8 Conclusion and Future Work

In this paper, we present PSPE, a Bayesian algorithm for the Pure exploration problem in episodic fixed-horizon MDPs. PSPE combines the Top-Two sampling procedure of PTS with PSRL. We define a notion of simple regret and show that it converges at an optimal exponential rate when using PSPE. Using stochastic chain MDPs, we compare the convergence of simple regret for PSPE with various values of parameter β. We also define the practical problem of Reinforcement Learning with practice. We empirically show that a combination of PSPE and PSRL can offer a feasible solution for this problem. We intend to further explore the problem of RL with practice and provide theoretical guarantees in the case of bandits and MDPs.

PSPE requires solving MDPs through dynamic programming at each step. An alternative approach, which avoids solving sampled MDPs is value function sampling [1]. Using value function sampling approaches to achieve pure exploration remains an open research direction.

References

[1] R. Dearden, N. Friedman, and S. Russell. Bayesian Q-learning. In *AAAI Conference on Artificial Intelligence*, pages 761–768, 1998.

[2] I. Osband, D. Russo, and B. Van Roy. (More) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, pages 3003–3011, 2013.

[3] I. Osband and B. Van Roy. Why is posterior sampling better than optimism for reinforcement learning. *arXiv preprint arXiv:1607.00215*, 2016.

[4] D. Russo. Simple bayesian algorithms for best arm identification. *Twenty ninth Annual Conference on Learning Theory*, pages 1417–1418, 2016.

[5] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Reinforcement Learning in Rich-Observation MDPs using Spectral Methods

Kamyar Azizzadenesheli [*]
Department of Electrical Eng. & Computer Sci.
University of California, Irvine
Irvine, CA 92697
kazizzad@uci.edu

Alessandro Lazaric[†]
INRIA, France
SequeL Team
INRIA Lille - Nord Europe
alessandro.lazaric@inria.fr

Animashree Anandkumar[‡]
Department of Electrical Eng. & Computer Sci.
University of California, Irvine
Irvine, CA 92697
a.anandkumar@uci.edu

Abstract

In this paper, we address the problem of online learning and decision-making in high-dimensional active dynamic environments where the agent is uncertain about the environment dynamics. The agent learns a policy in order to maximize a notion of payoff while her actions change the environment dynamics. We focus on the problem of learning in rich-observation Markov decision processes (ROMDP), where a low-dimensional MDP with X hidden states is observable through a possibly large number of observations. In ROMDPs, hidden states are mapped to observations through an injective mapping, so that an observation y can be generated by only one hidden state x, e.g., navigation problems, where the agent receives a sensory observation (high dimensional image) from the environment and needs to infer the current location (low dimensional hidden state) in order to make a decision. Due to the curse of dimensionality, ignoring the low dimensional latent structures results in an intolerable regret of $\widetilde{O}(Y\sqrt{AN})$ for well-known Reinforcement learning (RL) algorithm which is linear in a number of possible observations. Exploiting the latent structure, we devise a spectral learning method guaranteed to correctly reconstruct the mapping between hidden states and observations. We then integrate this method into UCRL (Upper Confidence bound RL) to obtain a reinforcement learning algorithm able to achieve a regret of order $\widetilde{O}(X\sqrt{AN})$ which matches the regret of UCRL and reaches and computation complexity of UCRL running directly on the hidden MDP.

Keywords: MDP, ROMDPs, UCRL, Spectral Methods.

Acknowledgements

K. Azizzadenesheli is supported in part by NSF Career Award CCF- 1254106. A. Lazaric is supported in part by the French Ministry of Higher Education and Research and the French National Research Agency (ANR) under project ExTra-Learn n.ANR-14-CE24-0010-01. A. Anandkumar is supported in part by Microsoft Faculty Fellowship, Google faculty award, Adobe grant, NSF Career Award CCF-1254106, ONR Award N00014-14-1-0665, and AFOSRYIP FA9550-15-1-0221.

---

[*] https://sites.google.com/uci.edu/kamyar

[†] http://chercheurs.lille.inria.fr/ lazaric/Webpage/Home/Home.html

[‡] https://newport.eecs.uci.edu/anandkumar/

1 Introduction

Reinforcement Learning (RL) is an effective approach to solving the problem of sequential decision–making under uncertainty. RL agents learn how to maximize long-term reward using the experience obtained by direct interaction with a stochastic environment. One of the main challenges in reinforcement learning (RL) [1] is how to properly trade off the *exploration* of an unknown environment and the *exploitation* of an estimate of the optimal policy to collect as much reward as possible over time. The exploration-exploitation trade-off has been vastly investigated under the PAC-MDP framework [2][3] and the regret minimization model [4][5], for which nearly-optimal algorithms are available. In particular, the UCRL [4] achieves a regret over N steps of order $\widetilde{O}(D_{\mathcal{Y}} Y \sqrt{AN})$[1] in environments with Y observable states, A actions, and diameter $D_{\mathcal{Y}}$ (i.e., the longest shortest path between any two states). Despite being nearly-optimal, in practice, the performance rapidly degrades as the number of states increases. Nonetheless, in many domains (e.g., robotics) the high-dimensional states observed by the agent are actually generated from an underlying low-dimensional latent space with $X \ll Y$ states that effectively summarize the MDP dynamics and rewards.

If the mapping between the latent states and the observations is known, then we could directly learn on the low-dimensional latent space and dramatically reduce the regret to $\widetilde{O}(D_{\mathcal{X}} X \sqrt{AN})$, where $D_{\mathcal{X}}$[2] is the diameter of the latent MDP. However, this mapping is typically unknown and needs to be learned by the agent. In this case, the main challenge is to efficiently learn the mapping so that the regret can scale with the number of latent states X and not the number of observations Y.

Summary of the results. In this paper, we show that for the class of rich-observation MDPs (ROMDP), this is indeed possible. In ROMDPs the mapping between hidden states to observations is injective, so that an observation can be generated by only one hidden state and both rewards and dynamics are direct functions of the hidden states. In other words, the hidden states form a non-overlapping clustering of observations and ROMDPs can be seen as a specific class of latent variable models (LVM). A number of tensor spectral decomposition methods have been developed to solve different LVMs, such as Gaussian mixture models, latent Dirichlet allocation, hidden Markov models, with strong sample-complexity guarantees (see e.g., Anandkumar et al. [6], Anandkumar et al. [7], Song et al. [8]).

In this paper, we build on this approach and we derive a novel spectral learning method based on the decomposition of the third moment of the observation process that allows us to correctly recover the mapping between hidden states and observations. In particular, we show that under the assumption of ergodicity and a technical assumption on the stochasticity of the ROMDP, the clustering returned by our method is correct with high probability. We then integrate this spectral method into the UCRL algorithm and we obtain an algorithm that trades off exploration and exploitation as the mapping from observations to hidden states is learned. We prove a regret bound of order $\widetilde{O}(D_{\mathcal{X}} X \sqrt{AN})$, thus matching the regret that can be obtained when the ROMDP structure is known, while the number of observations Y only appear in lower-order terms.

Related work. The existence of a latent space is an assumption that is commonly used in many online learning settings to reduce the learning complexity. In multi-armed bandit, Gheshlaghi-Azar et al. [9] and Maillard and Mannor [10] assume that a bandit problem is generated from an unknown (latent) finite set of problems. Gentile et al. [11] considers the more general scenario of latent contextual bandits, where the contexts belong to a few number of underlying hidden classes. Uniform exploration strategy over the contexts, combined with an online clustering algorithm is shown to achieve a regret scaling only with the number of hidden clusters. An extension to recommender systems is considered in [12] where the contexts type for both the users and items are unknown a priori. Again, uniform exploration is used, and the spectral algorithm of Anandkumar et al. [6] is deployed to learn the latent classes. The ROMDP model considered in this paper is a generalization of the latent contextual bandits, where actions influence the contexts (i.e., the states) and the objective is to maximize the long-term reward. ROMDPs have been studied by Krishnamurthy et al. [13], where a PAC analysis is given in the episodic framework and extended to the general case of contextual decision processes in [14]. Learning in ROMDPs can be also seen as a state-aggregation problem, where observations are aggregated to form a small latent MDP. While the literature on state-aggregation in RL is vast, most of the results have been derived for the batch setting (see e.g., Li et al. [15]). In online RL, Ortner [16] proposes a method to integrate state aggregation with UCRL. While the resulting algorithm may significantly reduce the computational complexity of extended value iteration, the analysis fails at showing any improvement in the regret.

Finally, we notice that ROMDPs are a special class of partially observable Markov decision processes (POMDP). Azizzadenesheli et al. [17] recently proposed an algorithm that leverages spectral learning methods to recover the mapping from states to observations, the reward, and the dynamics. While we follow a similar approach, learning in POMDPs is considerably more difficult than in ROMDPs and their final regret still scales with the number of observations. While the

---

[1] $D_{\mathcal{Y}} = \max_{y,y' \in \mathcal{Y}} \min_{\pi:\mathcal{Y}\to\mathcal{A}} \mathbb{E}\big[\tau_\pi(y,y')\big]$ is the diameter of the observation MDP with $\tau_\pi(y,y')$ being the (random) number of steps from y to y' by following the observation-based policy π

[2] $D_{\mathcal{X}} = \max_{x,x' \in \mathcal{X}} \min_{\pi:\mathcal{X}\to\mathcal{A}} \mathbb{E}\big[\tau_\pi(x,x')\big]$ is the diameter of the latent MDP with $\tau_\pi(X,x')$ being the (random) number of steps from x to x' by following the hidden state based policy π

computation of the optimal memoryless policy relies on an optimization oracle, which in general is NP-hard [19, 18][19]. On the other hand, computing the optimal policy in ROMDPs amounts to solving a standard MDP.

2 Preliminaries

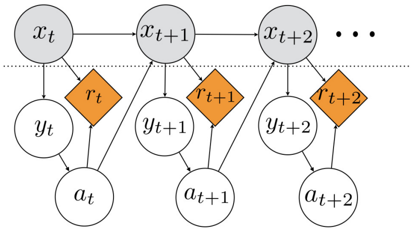


Figure 1: Graphical model of a ROMDP.

A rich-observation MDP (ROMDP) is a tuple $M = \langle \mathcal{X}, \mathcal{Y}, \mathcal{A}, R, f_T, f_O \rangle$, where $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{A}$ are the sets of hidden states, observations, and actions. We denote by X, Y, and A the cardinality of the sets, while their elements are enumerated by $i \in [X] = \{1..X\}$, $j \in [Y] = \{1..Y\}$, and $l \in [A] = \{1..A\}$. We assume that the number of hidden states is much smaller than the observations, i.e., $X \ll Y$. We assume the rewards to be bounded in $[0, 1]$ and to depend only on hidden states and actions, where the reward matrix $R \in \mathbb{R}^{A\times X}$ is such that $[R]_{i,l} = \mathbb{E}[r(x = i, a = l)]$. The dynamics of the MDP is defined directly on the hidden states as $T_{i',i,l} := f_T(i'|i, l) = \mathbb{P}(x' = i'|x = i, a = l)$, where we introduce the transition tensor $T \in \mathbb{R}^{X\times X\times A}$. Finally, the observations are generated as $[O]_{j,i} = f_O(j|i) = \mathbb{P}(y = j|x = i)$, where the observation matrix $O \in \mathbb{R}^{Y\times X}$ has minimum *non-zero* entry $O_{\min}$. The graphical model associated to a ROMDP is illustrated in Fig. 1.

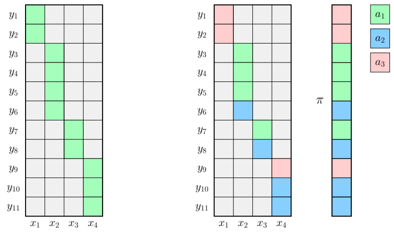


Figure 2: *(left)* Example of an observation matrix O. Since state and observation labelling is arbitrary, we arranged the non-zero values so as to display the block diagonal structure of the matrix. *(right)* Example of clustering that can be achieved by policy π.

While this model matches the general partially-observable MDP model, the observations of a ROMDP have a specific structure, where each observation y can be generated by one and only one hidden state. In other words, in ROMDPs the observation matrix O has a block structure where there is exactly one non-zero element in each row (see Fig. 2). This structure corresponds to assuming the existence of a non-overlapping clustering of the observations into X clusters/hidden states. We measure the performance of an observation-based policy $\pi_{\mathcal{Y}} : \mathcal{Y} \to \mathcal{A}$ starting from a hidden state x by its asymptotic regret at time N

$$R_N = N\rho^* - \left[\sum_{t=1}^{N} r_t\right], \quad \rho(x;\pi_{\mathcal{Y}}) = \lim_{n\to\infty} \mathbb{E}\left[\frac{\sum_{t=1}^{N} r_t}{N}\middle| x_0 = x, \pi_{\mathcal{Y}}\right].$$

where $\rho(x;\pi_{\mathcal{Y}})$ is asymptotic average reward and $\rho^* = \rho(\pi^*_{\mathcal{Y}})$. Our objective in the next section is to devise an algorithm whose regret tends to the regret attainable when the ROMDP structure is known in advance.

3 Stochastic latent MDP

In this section we introduce an algorithm combining UCRL with spectral tensor methods to recover the ROMDP representation. We first describe the methods that can be used to learn the structure of the observation matrix O, then we report our UCRL variant, and finally we derive its regret performance. In particular, we show that in order to improve the learning performance, we do not need to estimate the matrix O exactly, but only reconstruct the clusters $\{\mathcal{Y}_x\}_{x\in\mathcal{X}}$ correctly, which can be achieved by identifying the non-zero entries of O and not their exact value.

Assumptions. We focus on a set of ROMDPs satisfying the two following assumptions. For any policy $\pi_{\mathcal{Y}}$, the Markov chain induced on the hidden MDP M is ergodic and the transition matrix is full-rank.

Lemma 1. *Given a trajectory of N samples obtained by executing a policy π, for any action l, the spectral method can recover the mapping from observations, corresponding to action l, to hidden states (Emission matrix O) up to confidence bound $\mathcal{B}_O^{(l)} := C_2\sqrt{\frac{\log(2Y^{3/2}/\delta)}{N(l)}}$, with probability at least $1-\delta$, C_2 is a problem-dependent constant independent from the number of observations Y and $N(l) = \sum_{t=1}^{N} \mathbb{I}(a_t = l)$.*

The spectral method learns the structure of matrix O up to the confidence interval and partially reveals the structure of O matrix. Next, we enumerate over the observation set, cluster together the observations that w.h.p. belong to a same hidden state, $\widehat{\mathcal{Y}}_i$ enumerated with i, and construct a set of clusters $\widehat{\mathcal{Y}}$ with elements $\widehat{\mathcal{Y}}_i$. Consider a set $\widehat{\mathcal{Y}}^c = \mathcal{Y} \setminus \bigcup_{i,l} \widehat{\mathcal{Y}}_i$ with element $\widehat{\mathcal{Y}}^c_j$ enumerated j. Now construct a bigger set of $\mathcal{S}$, a union of set $\bigcup_i \widehat{\mathcal{Y}^c}_j$ and $\bigcup_i \widehat{\mathcal{Y}}_i$.

The algorithm. We now describe the spectral Learning UCRL (SL-UC) obtained by integrating the spectral methods with the UCRL strategy to optimize the exploration-exploitation trade-off and achieve small regret (Alg. 1), we initialize $\mathcal{S}^{(1)} = \mathcal{Y}$. The learning process is split into epochs of increasing length. At the beginning of each epoch $k > 1$, we first use the trajectory $(s_1, a_1, \ldots, s_{N^{(k)}})$, $s \in \mathcal{S}^{(k-1)}$ from the previous epoch to construct the auxiliary state space $\mathcal{S}^{(k)}$. At this point we can directly estimate the reward and transition model of the auxiliary MDP constructed on $\mathcal{S}^{(k)}$ by using standard empirical estimators.

$$\widehat{r}^{(k)}(s,a) = \frac{\sum_{t=1}^{N^{(k)}} r_t \mathbb{I}(s_t = s, a_t = a) r_t}{\max\{1, N^{(k)}(s,a)\}} \;\; ; \;\; \widehat{p}^{(k)}(s'|s,a) = \frac{N^{(k)}(s,a,s')}{\max\{1, N^{(k)}(s,a)\}}.$$

where $N^{(k)}(s,a) := \sum_{t=1}^{N^{(k)}} \mathbb{I}(s_t = s, a_t = a)$ and $N^{(k)}(s,a,s') := \sum_{t=1}^{N^{(k)}} \mathbb{I}(s_{t+1} = s', s_t = s, a_t = a)$ As a result, Chernoff-Hoeffding confidence bounds can be derived as in UCRL such that for any $s \in \widehat{\mathcal{S}}^{(k)}$ and $a \in \mathcal{A}$

$$\|p(\cdot|s,a) - \widehat{p}^{(k)}(\cdot|s,a)\|_1 \leq d_p(s,a) = \sqrt{\frac{7S^{(k)} \log(\frac{2AN^{(k)}}{\delta})}{\max\{1, N^{(k)}(s,a)\}}} \;\; , \;\; |\bar{r}(s,a) - \widehat{r}^{(k)}(s,a)| \leq d_r(s,a) = \sqrt{\frac{7 \log(\frac{2S^{(k)}AN^{(k)}}{\delta})}{2\max\{1, N^{(k)}(s,a)\}}}$$

with probability at least $1-\delta$, where $p(\cdot|s,a)$ and $\bar{r}$ are the transition probabilities and reward of the auxiliary MDP $M_{\mathcal{S}^{(k)}}$. At this point, among all the possible and plausible Aux-MDP models in the confidence intervals, we choose the optimistic model, which provides highest return. The resulting optimal policy of the optimistic model $\widetilde{\pi}^{(k)}$ is then executed until the number samples at least for one pair of auxiliary state and action is doubled. Moreover, to speed up the clustering we use the estimated reward functions and transition probabilities to leverage the clustering. Exploiting the the structure of reward function and transition probability on the top of spectral learning speeds up the general clustering, results in early stopping of time τ_0 and converges to latent MDP of size X.

Theorem 1 (Regret Bound of Stochastic Environment)**.** *Consider a ROMDP $M = \langle \mathcal{X}, \mathcal{Y}, \mathcal{A}, R, f_T, f_O \rangle$ characterized by a diameter $D_{\mathcal{X}}$ and $\tau_M = \max_\pi \max_x \mathbb{E}_\pi[\tau(x \to x)]$. If* SL-UC *is run over N time steps, it suffers the total regret of*

$$Reg_N \leq 34DX\sqrt{A(N-\tau)\log(N/\delta)} + \min\{\tau, 34D_Y Y\sqrt{A\tau \log(N/\delta)}\}$$

with probability at least $1-\delta$. The constant number of steps τ is the time that the algorithm takes to identify the whole mapping from observations to hidden states

$$\tau := \min\{C_1 16AY\tau_M(4C_2\frac{\log(Y/\delta)}{O_{\min}^2} + \log(XA/\delta)), \tau_0\}$$

where C_1 is a numerical constant which represents the similarity of observations in policy.

The bound reported above show that SL-UC achieves a regret roughly of the same order as if the hidden MDP was known from the beginning. On the other hand, it suffers from a constant term which depends on the observation MDP through Y and D_Y. The time to get to the right cluster τ also depends on the smallest probability of observation $O_{\min}$, since rare observations may take a long time before being correctly clustered. Notice that we left the additive term in the regret relatively implicit to stress the fact that the performance of SL-UC progressively gets better as more and more observations are clustered together since the number of aggregated $N^{(k)}(s,a)$ increases, the number of auxiliary states $\mathcal{S}^{(k)}$ decreases together with the diameter.

Algorithm 1 Spectral-Learning UCRL(SL-UC).

```
Input: Confidence $\delta'$
Initialize: $t = 1$, initial state $x_1$, $k = 1$, $\delta/N^6$
while $t < N$ do
    Run Spectral Methods with samples from epoch $k-1$ and obtain $\mathcal{S}$
    Compute the estimate reward $r^{(k)}$ and dynamics $p^{(k)}$
    Construct plausible set of AuxMDP $\mathcal{M}^{(k)}$ out of set $\mathcal{S}^k$
    Compute the optimistic policy
        $\widetilde{\pi}^{(k)} = \arg\max_{\pi} \max_{M \in \mathcal{M}^{(k)}} \rho(\pi; M)$    (1)
    Set $v^{(k)}(s,l) = 0$ for all actions $l \in \mathcal{A}$, $s \in \mathcal{S}^{(k)}$
    while $\forall l, \forall s, v^{(k)}(s,l) < \max\{1, N^{(k)}(s,l)\}$ do
        Execute $a_t = \widetilde{\pi}^{(k)}(s_t)$
        Observe reward $r_t$ and observation $y_t$
    end while
    Set $k = k + 1$
end while
```

4 Experiments

We validate our theoretical results by comparing the performance of SL-UC, UCRL2 (model based, exact solution) and DQN (model free, function approximation) [20], two well known RL algorithms. For DQN we implemented a three hidden-layers feed forward network (with no CNN block), equipped with RMSprop and reply buffer. We tuned the hyper parameters of the network and reported the best performance (achieved by network of size 30 × 30 × 30)). For SL-UC, in order to reveal the structure of emission matrix, we compute the confidence bound $\mathcal{B}_O$ via bootstrapping over samples.

We consider three randomly generated ROMDPs with $X = 5$, $A = 4$ and different $Y = 10, 20, 30$. Top to bottom in Fig. 3-left shows $Y = 10, 20, 30$ and reports the regret on a $\sqrt{N}$ scale. We see that in each instance the regret of UCRL and DQN grows much faster than SL-UC's, which leverages on the low-dimensional latent space to cluster observations and reduce the regret. Furthermore, while all regrets tend to be linear (i.e., growing as $\sqrt{N}$), we clearly see that the performance of UCRL and DQN is negatively affected by the increasing number of observations, while the regret of SL-UC stays almost constant, confirming that the hidden space $\mathcal{X}$ is learned very rapidly. In fact, we see from Fig. 3-middle that SL-UC is able to rapidly find the latent MDP. Furthermore we tweak SL-UC a little bit and test on randomly generated MDPs with no explicit hidden space. The objective is to verify whether SL-UC can be used to identify (approximate) clusters. As it is shown in Fig. 3-right, SL-UC is effective even in this scenario compared to UCRL and DQN.

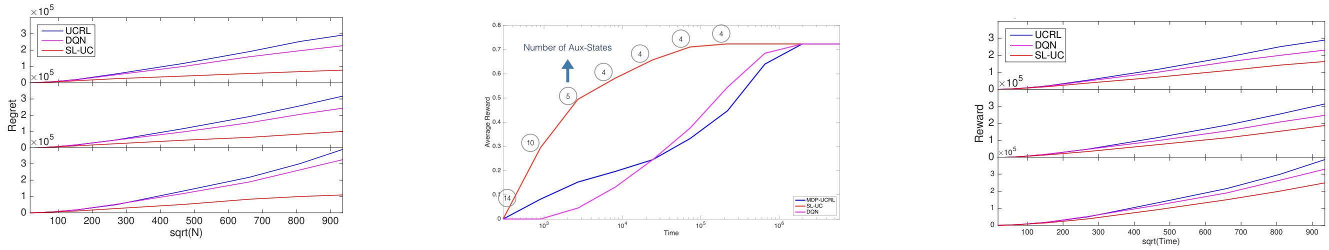


Figure 3: *(left)*Regret comparison for SL-UC in ROMDPs. *(Middle)* Learning rate of SL-UC. After first few epoch, it learns the the true mapping matrix. The numbers in the bulbs are the cardinality of Aux-MDP. *(right)* The regret comparison for SL-UC in randomly generated MDP, $A = 4$, from top to bottom $Y = 10, 20, 30$,

5 Conclusion

We introduced SL-UC, a novel RL algorithm to learn in ROMDPs combining a spectral method for recovering the clustering structure of the problem and UCRL to effectively trade off exploration and exploitation. We proved theoretical guarantees showing that SL-UC progressively refines the clustering so that its regret tends to the regret that could be achieved when the hidden structure is known in advance. For the deterministic ROMDPs, in fact, despite not being ergodic, in this case we can first uniformly explore the latent space and collect sufficient number of sample to find the exact clustering and reduce the large MDP to the latent MDP and then apply UCRL on the latent MDP. These two-phase algorithm would suffer a constant regret of pure exploration at the beginning and regret of $\widetilde{\mathcal{O}}(D_{\mathcal{X}}X\sqrt{AN})$ due to UCRL in the second phase.

References

[1] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[2] Alexander L. Strehl, Lihong Li, and Michael L. Littman. Reinforcement learning in finite mdps: Pac analysis. *J. Mach. Learn. Res.*, 10:2413–2444, 2009.

[3] Zhaohan Daniel Guo, Shayan Doroudi, and Emma Brunskill. A pac rl algorithm for episodic pomdps. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 510–518, 2016.

[4] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

[5] Peter L. Bartlett and Ambuj Tewari. REGAL: A regularization based algorithm for reinforcement learning in weakly communicating MDPs. In *Proceedings of the 25th Annual Conference on Uncertainty in Artificial Intelligence*, 2009.

[6] Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15(1):2773–2832, 2014.

[7] Animashree Anandkumar, Daniel Hsu, and Sham M Kakade. A method of moments for mixture models and hidden markov models. *arXiv preprint arXiv:1203.0683*, 2012.

[8] Le Song, Animashree Anandkumar, Bo Dai, and Bo Xie. Nonparametric estimation of multi-view latent variable models. *arXiv preprint arXiv:1311.3287*, 2013.

[9] Mohammad Gheshlaghi-Azar, Alessandro Lazaric, and Emma Brunskill. Sequential transfer in multi-armed bandit with finite set of models. In *Advances in Neural Information Processing Systems 26*, pages 2220–2228. 2013.

[10] Odalric-Ambrym Maillard and Shie Mannor. Latent bandits. In *Proceedings of the Thirty-First International Conference on Machine Learning (ICML'14)*, 2014.

[11] Claudio Gentile, Shuai Li, and Giovanni Zappella. Online clustering of bandits. In *ICML*, pages 757–765, 2014.

[12] Aditya Gopalan, Odalric-Ambrym Maillard, and Mohammadi Zaki. Low-rank bandits with latent mixtures. *arXiv preprint arXiv:1609.01508*, 2016.

[13] Akshay Krishnamurthy, Alekh Agarwal, and John Langford. Pac reinforcement learning with rich observations. In *Advances in Neural Information Processing Systems*, pages 1840–1848, 2016.

[14] Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E Schapire. Contextual decision processes with low bellman rank are pac-learnable. *arXiv preprint arXiv:1610.09512*, 2016.

[15] Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for mdps. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics (ISAIM-06)*, 2006.

[16] Ronald Ortner. Adaptive aggregation for reinforcement learning in average reward markov decision processes. *Annals of Operations Research*, 208(1):321–336, 2013.

[17] Kamyar Azizzadenesheli, Alessandro Lazaric, and Animashree Anandkumar. Reinforcement learning of pomdps using spectral methods. In *Proceedings of the 29th Annual Conference on Learning Theory (COLT)*, 2016.

[18] Michael L. Littman. Memoryless policies: Theoretical limitations and practical results. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior : From Animals to Animats 3: From Animals to Animats 3*, SAB94, pages 238–245, Cambridge, MA, USA, 1994. MIT Press. ISBN 0-262-53122-4.

[19] Kamyar Azizzadenesheli, Alessandro Lazaric, and Animashree Anandkumar. Open problem: Approximate planning of pomdps in the class of memoryless policies. *arXiv preprint arXiv:1608.04996*, 2016.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Communications that Emerge through Reinforcement Learning Using a (Recurrent) Neural Network

Katsunari Shibata[*]
Department of Innovative Engineering
Oita University
700 Dannoharu, Oita 870-1192, JAPAN
`katsunarishibata@gmail.com`

Abstract

Communication is not only an action of choosing a signal, but needs to consider the context and the sensor signals. It also needs to decide what information is communicated and how it is represented in or understood from signals. Therefore, **communication should be realized comprehensively together with its purpose and other functions**.

The recent successful results in end-to-end reinforcement learning (RL) show the importance of comprehensive learning and the usefulness of end-to-end RL for it. Although little is known, the author's group has shown that a variety of communications emerge through RL using a (recurrent) neural network (NN). Here, three of our works are introduced again for the coming leap in this field.

In the 1st one, **negotiation to avoid conflicts** among 4 randomly-picked agents was learned. Each agent generates a binary signal from the output of its recurrent NN (RNN), and receives 4 signals from the agents three times. After learning, each agent successfully made an appropriate final decision after negotiation for any combination of 4 agents. **Differentiation of individuality** among the agents also could be seen.

The 2nd one focused on **discretization of communication signal**. A sender agent perceives the receiver's location and generates a continuous signal twice by its RNN. A receiver agent receives them sequentially, and moves according to its RNN's output to reach the sender's location. When noises were added to the signal, it was binarized through learning and **2-bit communication** was established.

The 3rd one focused on **end-to-end comprehensive communication**. A sender receives **1,785-pixel real camera image** on which a real robot can be seen, and sends two sounds whose frequencies are computed by its NN. A receiver receives them, and **two motion commands for the robot** are generated by its NN. After learning, though some preliminary learning was necessary for the sender, the robot could reach the goal successfully from any initial location.

Keywords: reinforcement learning (RL), (recurrent) neural network (RNN), negotiation, signal discretization, grounding communication

Acknowledgements

This research has been supported by JSPS KAKENHI Grant Numbers JP15300064, JP19300070, JP23500245 and many our group members.

---

[*] http://shws.cc.oita-u.ac.jp/˜shibata/home.html

1 Introduction

Needless to say that communication is a kind of higher functions for us, and how language and communication emerge has been discussed actively[1]. Especially, establishment of communication protocol, discretization of communication signals, realization of dynamic communication and also grounding of communication have been open problems for a long time. The origin of the research can be seen in [2, 3, 4, 5], and many works have been done as in[6, 7]. However, in most of them, existence of symbols or some special architecture for communication is presupposed, and the communication is dealt with separately from the other functions or its original purpose. However, communication needs not only to choose a signal from possible ones, but needs to consider the context and the present sensor signals. It also needs to decide what information should be communicated and how they should be represented in or understood from signals to achieve some purpose. It is a really comprehensive function actually.

On the other hand, the impressive results by Google DeepMind[8, 9] make us notice the importance of end-to-end comprehensive learning and usefulness of reinforcement learning (RL) using a neural network (NN) for it. A challenge on this basis can be seen also in the field of communication learning[1].

However, actually, although little is known to the public unfortunately, our group has propounded the importance of end-to-end RL for the emergence of comprehensive function for more than 20 years[10], and has shown already several works also in the emergence of communication. In this paper, they are introduced at this timing to contribute the coming leap in this research field hereafter.

Before entering the introduction of our works, our basic standpoint or setting for communication learning is explained at first. There are two or more learning agents each of which has its own (recurrent) NN. Each communication signal is generated based on the output of an agent's NN, and becomes the input of one or some other agents' NNs. The agents learn independently based on reward or punishment without any communication for learning, that means learning is completely decentralized. No knowledge about what information should be communicated is given at all. A reward or punishment is given only at the end of one episode, and it is not given directly for the communication signal, but for the result of one or a series of motions each of which is done based on the communication.

The emergence of three types of communication are introduced here. In all of them, communication protocol was acquired through learning. In the 1st one, negotiation to avoid conflicts among multiple agents that needs dynamical communication emerged through learning using a recurrent neural network (RNN)[11]. In the 2nd one, communication signal was binarized through learning in a noisy environment[12]. In the 3rd one, grounded communication emerged in which a sender generated communication signals from raw camera pixel image, and a receiver received them and generated motion commands for a real robot[13]. The detail of each work can be found in each reference.

2 Learning of Negotiation to Avoid Conflicts (1999)[11]

At first, it is shown that as dynamic communication, negotiation among multiple agents emerges through learning only from rewards and punishments. As shown in Fig. 1, there are 8 agents and 4 of them are picked up randomly at each episode. Each of 4 agents is located randomly at one of the entrances of 4 paths. 3 communication chances are given to the agents, and after that as the final decision, each agent decides which of the two paths, ○ and ▽, it will go to. At each chance, each agent can send a binary signal and receive 4 signals from the other 3 agents and itself as the inputs for the next time step. If neighbor agents choose the same path, they collide and a penalty is given to the both agents. If an agent can go through the path without collision, the agent can get a reward, but does not share it with the other agents. If all the agents choose the path other than their two neighbors and the same as its opposite, they can rotate clockwise or anti-clockwise without any collision. The agent cannot know either the source of each signal or which agents are chosen.

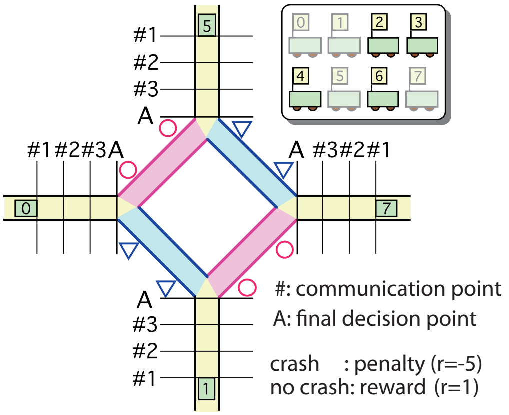


Figure 1: Negotiation task. 4 agents are picked up randomly. They negotiate through 3 chances and decide which path to go finally to avoid a collision with a neighbor agent. (1999)[11]

Each agent has its own RNN. As shown in Fig. 2, the RNN has two outputs, one of which is used to decide its communication signal and the other is used to decide its action. The communication signal or action is stochastically chosen from -1 and 1. The probability for taking the value 1 is decided by the output of the RNN. The first communication signal is not depending on the other agents' signals and always the same after learning. Here, a primitive type of RL is employed. When each agent receives a reward, the probabilities of the action and a series of the communication signals are learned to be large, and when it receives a penalty, the probabilities are learned to be small. The RNN is trained by BPTT (Back Propagation Through Time) learning[14]. It is not so easy for us to design the communication strategy of each agent, but all the agents went through a path successfully for any combination of agents after learning.

Two examples of negotiation are shown in Tab. 1. In the case of Table 1 (a), agents 0, 2, 5, 7 were chosen. At the 1st communication chance #1, only the signal from the agent 5 was 1. At the chance #2, the agent 0 changed the signal to 1, and at the chance #3, all the signals were the same as the chance #2. The final decision of each agent was the opposite of the 3rd signal. By observing many cases, it could be interpreted that the signal 1 means the request to go to the path ▽ (action -1). In this case, if only the agent 0 changed its mind, then no conflict occurred. The agent 0 seems to have understood the situation from the signals at the communication chance #1 and changed its decision flexibly at the chance #2.

In the case of Table 1 (b), agents 0, 2, 3, 7 were chosen. The first and second communication signals for 4 agents were the same, and no one gave up their preferences. However, at the communication chance #3, the agent 7 changed its signal even though the inputs of the RNN were completely the same as the chance #2. Then, the agent 2 that was located at the opposite side of the agent 7 understood the situation and changed the final action, and then the agent 0 and 3 took the action 1 (path ○) and the others took the action -1 (path ▽).

The interesting thing is that though the RNN was the same among the agents (the initial weights were set randomly) and learning was completely independent, the individuality emerged through learning. The most stubborn agent hardly gave up its preference, and the most indecisive agent changed its mind soon when some conflict occurred.

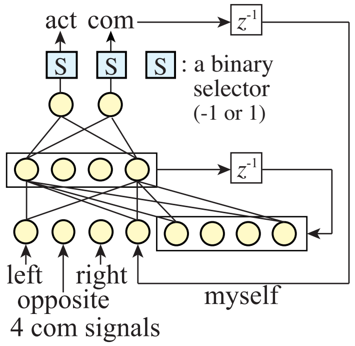


Figure 2: A recurrent neural network (RNN) for each agent.[11]

Table 1: Two examples of negotiation after learning.[11]

	#1	#2	#3	Act
Agent 0	-1	1	1	-1 ▽
Agent 2	-1	-1	-1	1 ○
Agent 5	1	1	1	-1 ▽
Agent 7	-1	-1	-1	1 ○

(a)

	#1	#2	#3	Act
Agent 0	-1	-1	-1	1 ○
Agent 2	-1	-1	-1	-1 ▽
Agent 3	-1	-1	-1	1 ○
Agent 7	-1	-1	1	-1 ▽

(b)

com_signal = –1 → request for the path ○ (Act = 1)

🟨 : cannot be realized without an RNN

3 Emergence of Signal Discretization (2005)[12]

We use language or symbols for communication. The symbols can be considered as the result of discretization of communication signals. Therefore, if communication signals are discretized through learning, it is very interesting from the viewpoint of solving the mystery of symbol emergence.

A simple reason for the emergence of signal discretization that is thought of easily is "the need for correct information transmission even in a noisy environment". In [1], possibility of the emergence was suggested from the result of centralized learning in a special structure. Before that, the author examined discretization of continuous-valued communication signals from a need through decentralized reinforcement learning (RL) as follows. The details can be found in [12].

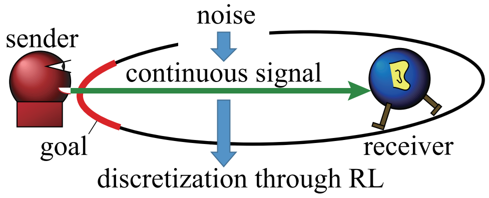


Figure 3: A one-way communication task to examine the binarization of continuous-valued signals by noise addition through learning.[12]

A very simple one-way communication task is assumed. As shown in Fig. 3, there are two agents, which are a sender and a receiver, and they exist in a one dimensional ring-shape space. Each of them has their own recurrent neural network (RNN), and learns to communicate. In each RNN, there are two outputs, actor and critic, as shown in Fig. 4. The sender has a sensor to perceive the receiver's location, but cannot move. It receives vision-like 30 sensor signals each of which responds the existence of receiver agent in a local area of the space, and put the signals to its own RNN as inputs directly. The actor output of the RNN is sent directly as a continuous communication signal to the receiver. The receiver receives the signal, puts it to its own RNN as input, and computes its RNN. At the next time step, the sender computes its RNN, and the new output (signal) is sent to the receiver again. The receiver puts the signal to its RNN as input, and computes its RNN again. After that, the receiver moves according to the actor output of its RNN. When the receiver reached the goal close to the sender without overshooting, they can both get a reward, and learns independently. The task is very easy such that if the receiver takes appropriate motion, it can reach the goal in one step from any place in the space. The both RNNs are learned using the training signals generated based on the popular TD-type RL for actor-critic. However, the actor output is not used as the probability for a discrete action, but used directly as a continuous communication signal from the sender or continuous motion in the receiver.

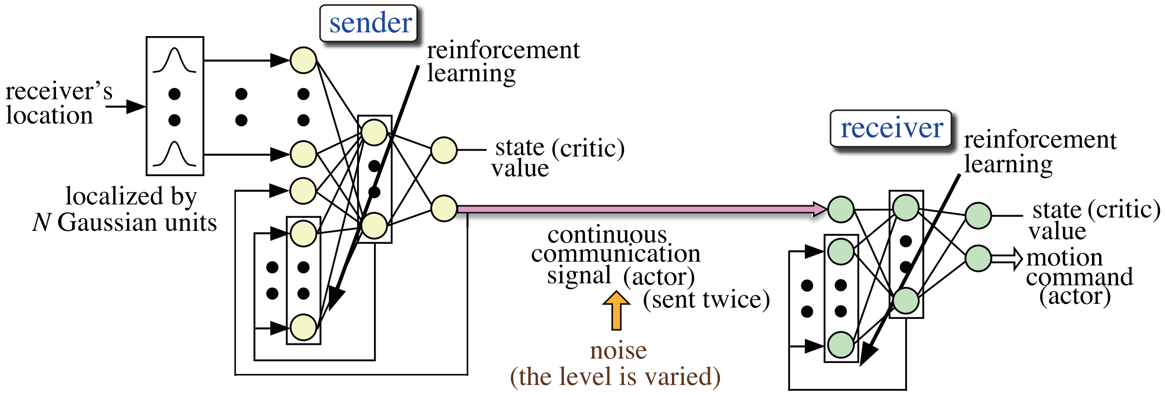
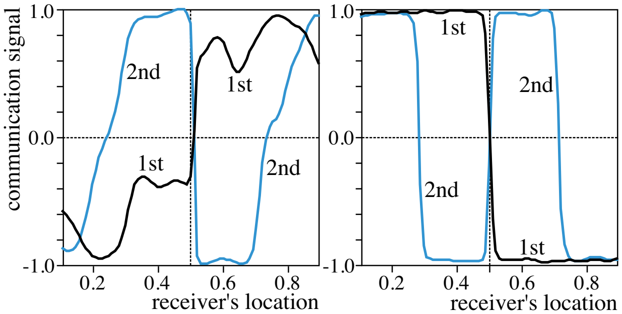


Figure 4: Signal flow in the one-way communication task. Each of the sender and the receiver has its own RNN. Continuous-valued communication signal that is the actor output of the sender's RNN is given to the receiver's RNN directly as inputs. The receiver moves according to its actor output after receiving two signals sequentially from the sender. Change in the signal representation by the noise addition is observed.(2005)[12]

After learning, the communication was established, and the receiver could reach the goal in one step from any location. Even though what information should be communicated was not given beforehand, it could be acquired from the necessity only through completely independent learning. It is also interesting that the receiver's location could be transmitted using the two signals that were sent sequentially.

Furthermore, noises were added to the signal during the learning phase. Fig. 5 shows the two signals as a function of the receiver's location after learning for the two cases of no noise and addition of some noise. The communication signals after learning with no noise seem irregular, but the receiver could reach the goal in one step. However, it was difficult to reach the goal in one step in the noisy environment. On the other hand, when learning in the noisy environment, the sender outputs became almost binary and 2-bit communication was established. In this task, to reach the goal in one step, the receiver was required to know its rough location. Then the sender learned to send which of the four areas in the space the receiver existed using two bit binary signals, and the receiver learned to interpret the two bit binary signals and moved appropriately to the goal in one step. It is surprising that even though the output of the RNN was a continuous value, and no directions or no options of "discretization" were given, they discovered 2-bit communication by themselves from the purpose of successful goal reaching for the receiver. The author does not think that the discrete fashion of language or symbols can be explained only by the above, but could show the possibility that they emerge through purposive learning though the dynamics of an RNN should be utilized further.



(a) noise_level = 0.0
(b) noise_level = 1.0

Figure 5: Communication signals after learning. (a) no noise is added and (b) some noises are added to the signals. The number in the graph indicates the order of the two sequentially transmitted signals. In (b), 2-bit communication was established without any direction.[12]

4 Learning of Purposive and Grounded Communication (2011)[13]

Grounding symbol or communication has been a serious problem in AI for a long time[15, 16]. Just manipulation of symbols cannot be called communication, but they should reflect the state of the other agents, objects or environment that can be perceived through sensors. Aiming to solve the problem, emergence of lexicon was shown[2, 16]. However, they employed special learning for the emergence under a special assumption of sharing the same observation. Learning of communication should be realized comprehensively with its purpose and other functions. Then our group showed that through the general end-to-end RL (actor-critic), communication in which signals are generated from a real camera color image as raw sensor signals emerged for the purpose of leading a real robot to its goal.

As shown in Fig. 6, a sender agent receives a color image from the camera that captures the robot at its center from the above. The image consists of 26x20 pixels originally, but the resolution of the 5x5 center area is doubled. Then 1,785 raw camera pixel image signals are the input of the sender's neural network (NN). Two sounds whose frequencies are decided by the two actor outputs are sent from a real speaker sequentially. A receiver receives the two sounds

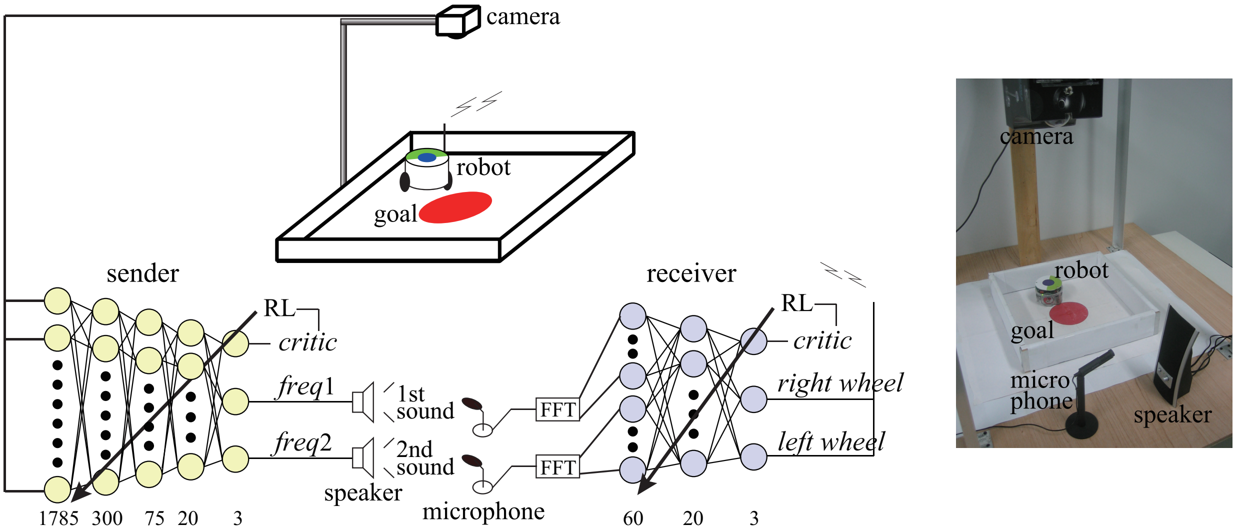


Figure 6: Emergence of communication through end-to-end reinforcement learning for the purpose of leading a real robot to the red goal area. The sound signals are generated from raw image signals from a real camera. (2011)[13]

using a real microphone, and uses them as inputs of its NN after FFT. The NN outputs the rotation angle for the two wheels of the robot and they are sent to the robot via bluetooth wireless communication. They learn independently from the reward that is given when the robot reaches a given goal (red circle area). However, it was difficult to establish the communication through learning from scratch, then the sender learned to move the robot directly from the images before this communication learning. By using the weights in the hidden neurons after the preliminary learning as initial weights for the communication learning in the sender agent, communication was established and the robot could reach the goal after learning. It is the world's first establishment of communication through end-to-end RL using a real robot and camera as long as the author knows.

References

[1] Foerster, J. N., Assael, Y. M., de Freitas, N. and Whiteson, S. (2016) Learning to Communicate with Deep Multi-Agent Reinforcement Learning, *arXiv*:1605.06676v2.

[2] Nakano, N., Sakaguchi, Y., Isotani, R. & Ohmori, T. (1988) Self-Organizing System Obtaining Communication Ability, *Biological Cybernetics*, **58**, 417–425.

[3] Werner, G. M. & Dyer, M. G. (1991) Evolution of Communication in Artificial Organisms, *Proc. of Artificial Life* II, 1–47

[4] Davis, R. & Smith, R. G. (1983) Negotiation as a Metaphor for Distributed Problem Solving, *Artificial Intelligence*, **20**(1), 63–109.

[5] Kreifelts, T. & von Martial, F. (1991) A Negotiation Framework for Autonomous Agents, Demazeau, Y. & Muller, J.-P. eds., *Decentralized A. I.* 2, 71–88.

[6] Lyon, C., Nahaniv, C. L. & Cangelosi, A. (2007) Emergence of Communication and Language, Springer.

[7] Steels, L. (2015) The Talking Heads experiment: Origins of words and meanings, Berlin, Language Science Press.

[8] Minh, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning *Nature*, **518**, pp. 529–533.

[9] Silver, D., Huang, A., et al. (2016). Mastering the game of Go with deep neural networks and tree search, *Nature*, **529**, 484–489.

[10] Shibata, K. (2017). Functions that Emerge through End-to-End Reinforcement Learning, *RLDM 2017*.

[11] Shibata, K. & Ito, K (1999) Emergence of Communication for Negotiation By a Recurrent Neural Network, *Proc. of ISADS (Int'l Symposium on Autonomous Decentralized System) '99*, 294-301.

[12] Shibata, K. (2005). Discretization of Series of Communication Signals in Noisy Environment by Reinforcement Learning, In Ribeiro et al., *Adaptive and Natural Computing Algorithms (ICANNGA '05)*, 486–489.

[13] Shibata, K. & Sasahara, K. (2011). Emergence of Purposive and Grounded Communication through Reinforcement Learning, *Adv. in Neuro-Information Processing, LNCS (ICONIP2011)*, **7064**, 66–75.

[14] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Internal Representations by Error Propagation, In Rumelhart, D. E., & McClelland, J. L., *Parallel Distributed Processing*, **I**, 318–362. Bradford Book. Cambridge, MA: MIT Press.

[15] Harnad, S. (1990). The Symbol Grounding Problem, *Physica D*, **42**, 335–346.

[16] Steels, L. (2003) Evolving grounded communication for robots, *Trends in Cognitive Science*, **7**(7), 308–312

Functions that Emerge through End-to-End Reinforcement Learning — The Direction for Artificial General Intelligence —

Katsunari Shibata[*]
Department of Innovative Engineering
Oita University
700 Dannoharu, Oita 870-1192, JAPAN
`katsunarishibata@gmail.com`

Abstract

Recently, triggered by the impressive results in TV-games or game of Go by Google DeepMind, end-to-end reinforcement learning (RL) is collecting attentions. Although little is known, the author's group has propounded this framework for around 20 years and already has shown a variety of functions that emerge in a neural network (NN) through RL. In this paper, they are introduced again at this timing.

"Function Modularization" approach is deeply penetrated subconsciously. The inputs and outputs for a learning system can be raw sensor signals and motor commands. "State space" or "action space" generally used in RL show the existence of functional modules. That has limited reinforcement learning to learning only for the action-planning module. In order to extend reinforcement learning to learning of the entire function on a huge degree of freedom of a massively parallel learning system and to explain or develop human-like intelligence, the author has believed that end-to-end RL from sensors to motors using a recurrent NN (RNN) becomes an essential key. Especially in the higher functions, since their inputs or outputs are difficult to decide, this approach is very effective by being free from the need to decide them.

The functions that emerge, we have confirmed, through RL using a NN cover a broad range from real robot learning with raw camera pixel inputs to acquisition of dynamic functions in a RNN. Those are (1)image recognition, (2)color constancy (optical illusion), (3)sensor motion (active recognition), (4)hand-eye coordination and hand reaching movement, (5)explanation of brain activities, (6)communication, (7)knowledge transfer, (8)memory, (9)selective attention, (10)prediction, (11)exploration. The end-to-end RL enables the emergence of very flexible comprehensive functions that consider many things in parallel although it is difficult to give the boundary of each function clearly.

Keywords: function emergence, end-to-end reinforcement learning (RL), recurrent neural network (RNN), higher functions, artificial general intelligence (AGI)

Acknowledgements

This research has been supported by JSPS KAKENHI Grant Numbers JP07780305, JP08233204, JP13780295, JP15300064, JP19300070 and many our group members

---

[*] http://shws.cc.oita-u.ac.jp/~shibata/home.html

1 Introduction

Recently, triggered by the impressive results in TV-games[1, 2] or game of Go[3] by Google DeepMind, the ability of reinforcement learning (RL) using a neural network (NN) and the importance of end-to-end RL is collecting attentions. One remarkable point especially in the results in TV-games is the gap such that even though the inputs of a deep NN are raw image pixels without any pre-processing and the NN is just learned through RL, the ability acquired through learning extends to the excellent strategies for several games. The learning do not need special knowledge about learned tasks and necessary functions emerge through learning, and so it is strongly expected to open the way to the Artificial General Intelligence (AGI) or Strong AI.

It has been general that a NN is considered as just a non-linear function approximator for RL, and a recurrent neural network (RNN) is used to avoid POMDP (Partially Observable Markov Decision Problem). Under such circumstances, the origin of the end-to-end RL can be found in the Tesauro's work called TD-gammon[4]. The author's group is the only one who has propounded this framework consistently for around 20 years using sometimes the symbolic name of "Direct-Vision-based Reinforcement Learning"[7, 8] and has shown already a variety of functions that emerge in a NN or RNN through RL[9] although little is known about them unfortunately.

In this paper, the author's unwavering direction that end-to-end RL becomes an important key for explaining human intelligence or developing human-like intelligence especially for the higher functions is introduced at first. It is also shown that a variety of functions emerge through end-to-end (oriented) RL; from real robot learning with raw camera pixel inputs to acquisition of dynamic functions in an RNN. All of the works here have been published already, but the author believes that it is worthwhile to know what functions emerge and what functions hardly emerge at this timing when the end-to-end RL begins to be focused on.

2 The Direction for Human-like General Intelligence

There is probably little doubt that human intelligence is realized thanks to the massively parallel and cohesively flexible processing on a huge degree of freedom in our brain. On the other hand, unlike in the case of unconsciousness, our consciousness looks linguistic and so it is not parallel but sequential. Therefore, it is impossible to completely understand the brain functions through our consciousness. Nevertheless, the researchers have tried to understand the brain or develop human-like intelligence by hands. We are likely to divide a difficult problem into sub-problems expecting each divided one is easier to solve. Then "Function Modularization" approach has been deeply penetrated subconsciously. However, for each module, we have to decide what are the inputs and outputs at first. It is easily known that to decide the information that comes and goes between divided modules, it is necessary to understand the entire function. Then to decide the information, some simple frame is set in advance and that causes the "Frame Problem". It can be also thought that the division into "symbolic (logical) processing" and "pattern processing" causes the "Symbol Grounding Problem". In our brain, they may be processed in different areas, but they must be closely related in the same brain as one NN.

"State space" or "action space", which is generally used in RL, can be another aspect of the function modularization. Researchers have limited the learning only to the actions that make the mapping from state space to action space, and have tried to develop the way of construction of state space from sensor signals and given the way of generating motor commands from each action separately from RL. It has also been taken for granted that in recognition problems, classification categories are given and in control problems, reference trajectories are given in advance usually. However, we can recognize complicated situations from a picture, but it is clear that all of them cannot be given as classification targets in advance. There is no evidence that reference trajectory is explicitly generated in our brain. From the viewpoint of understanding and developing human-like general intelligence, they are by-products of the function modularization. They give unnecessary constraints on a huge degree of freedom, and disturb the flexible and comprehensive learning despite the intension of the researchers.

Based on the above, the author has thought that the interruption of human designers should be cut off and the development of functions should be committed to learning of high-dimensional parallel systems as much as possible. The inputs for a learning system can be raw sensor signals, the outputs can be raw actuator signals, and the entire process from sensors to actuators (motors) should be learned without dividing into functional modules. A NN has an ability to optimize the parallel process according to some value or cost function through learning. If training signals are given by humans, they can be constraints for learning. However, RL has an ability to learn autonomously without receiving training signals directly based on trials and errors using a motion-perception feedback loop with the environment.

For the higher functions, which are different from the recognition or control, it is difficult to decide either inputs or outputs, and that has disturbed the progress of the research on them. The author is expecting that higher functions also emerge through comprehensive end-to-end RL from sensors to motors using a recurrent NN. In image recognition, the use of deep learning makes the performance better than the conventional approaches that need to design a feature space[5]. In speech recognition, to leave the learning to an RNN makes the performance better than the combination with conventional methods like HMM[6]. They seem to support the importance of end-to-end learning approach.

The author has thought that what emerges should be called "functions" rather than "representation" because it is not just a choice of representation, but the result of the processing to get there. Furthermore to be more accurate, the function that emerges cannot be localized clearly in the NN, but it is just a label for us to understand through our consciousness.

3 Functions that Emerge through Reinforcement Learning (RL)

Here, functions that have been observed to emerge in a NN through RL in our works are introduced. The NN is trained using the training signals produced automatically by RL; Q-learning, actor-critic or Actor-Q (for learning of both continuous motion and discrete action) based on TD learning. By using actor outputs in actor-critic or actor-Q directly as motion commands, not probability called policy but continuous motions have been learned directly. The structure like convolutional NN is not used except for the work in [22]. The details of each work can be found in each reference.

3.1 Static Functions that Emerge in a Layered Neural Network (NN)

A layered neural network (NN) is used and trained according to error back-propagation (BP), and static functions emerge as follows.

§ **Image Recognition**
Same as [1, 2], images (pixels) were put into a neural network directly as inputs, and appropriate behaviors to get a reward were acquired[7, 8]. That was confirmed also in real robot tasks; Box Pushing (continuous motion)[10] and Kissing AIBO (real-world-like environment)[11, 12] as shown in Fig. 1 and Fig. 2.

§ **Color Constancy (Optical Illusion)**
Motion control of a colored object to the goal location decided by the object color was learned. The top view image covered by a randomly-appearing colored filter was the input. From the internal representation, we tried to explain the optical illusion of color constancy[13].

§ **Sensor Motion (Active Recognition)**
A pattern image was the input, and from the reward that indicates whether the recognition result is correct or not, both recognition and camera motion for better recognition were acquired through RL[14].

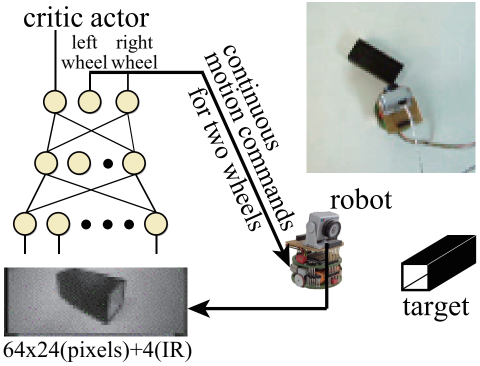


Figure 1: Learning of Box Pushing: End-to-end reinforcement learning (RL) using a real robot (Khepera). A reward was given only when the robot pushed the box, and continuous motions for the two wheels were learned. (2003)[10]

§ **Hand-Eye Coordination and Hand Reaching Movement**
A NN whose inputs were joint angles of an arm and also image on which its hand can be seen, and whose outputs were the joint torques learned to reach the hand to the randomly-located target that could be also seen on the image. No explicit reference trajectory was given. Furthermore, adaptation of force field and its after effect were observed[15].

§ **Explanation of the Brain Activations during Tool Use**
From the internal representation after learning of a reaching task with variable link length, we tried to explain the emergence of the activities observed in the monkey brain when the monkey used a tool to get a food[16].

§ **Knowledge Transfer between Different Sensors**
An agent has two kinds of sensors and two kinds of motors. There are four sensor-and-motor combinations. There is a task that could be achieved using either of the combinations. After the agent learned the task using 3 combinations, learning for the remainder sensor-and-motor combination was drastically accelerated[17].

§ **Game Strategy** (Not our works, but wonderful results can be seen in [4, 1, 2, 3])

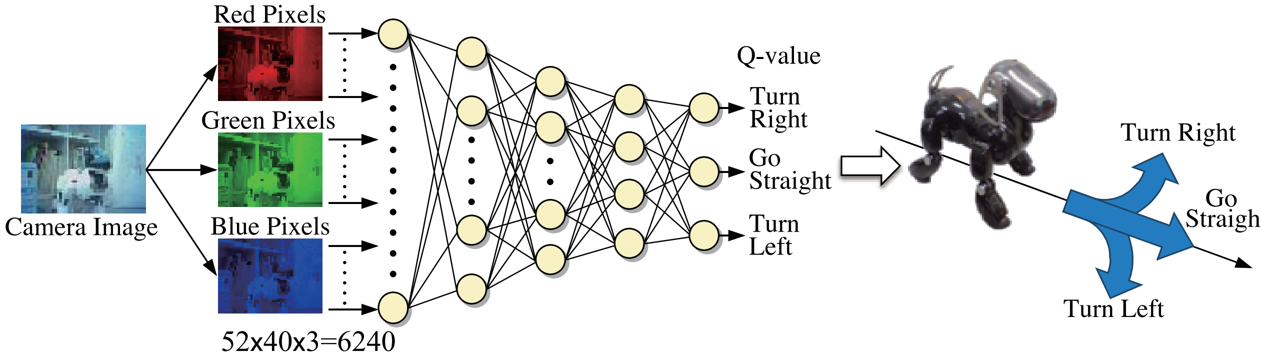


Figure 2: Learning of approaching to kiss the other AIBO (not moving): a real robot (AIBO) learned using a DQN (Deep-Q Network) (convolutional structure was not used) in a real-world-like environment. (2008)[11]

3.2 Dynamic Functions that Emerge in a Recurrent Neural Network (RNN)

Here, the emergence of dynamic functions is introduced in which expansion along the time axis is required. In this case, an RNN is used to deal with dynamics, and is trained by BPTT(Back Propagation Through Time) with the training signals generated automatically based on RL. For both acquisition of memory and error propagation, the feedback connection weights are set so that the transition matrix is the identity matrix or close to it when being linearly approximated.

§ **Memory**
There are some works in which necessary information was extracted, memorized and reflected to behaviors after learning almost only from a reward at each goal and punishment. In [18], a very interesting behavior in which if unexpected results occurred, an agent went back to check the state in the previous stage without any direction could be observed. In [19] and [20], a real camera image was used as input, and both camera motion and pattern meaning[19] or both camera motion and word recognition[20] were learned. Purposive associative memory could be also observed[19].

§ **Selective Attention**
It was learned that in a task in which the attended area of the next presented pattern is changed according to the previously-presented image, the area of the image was correctly classified without any special structure for attention[21]. Here, TD-based RL was not used, but learning was done by the reinforcement signal representing only whether the final recognition result was correct or not. Purposive associative memory could be also observed.

§ **Prediction** (shown in the next subsection[22])

§ **Explanation of the Emergence of Reward Expectancy Neurons**
From a simulation result of RL using an RNN, we tried to explain the emergence of reward expectancy neurons, which responded only in the non-reward trials in a multi-trial task observed in the monkey brain[23].

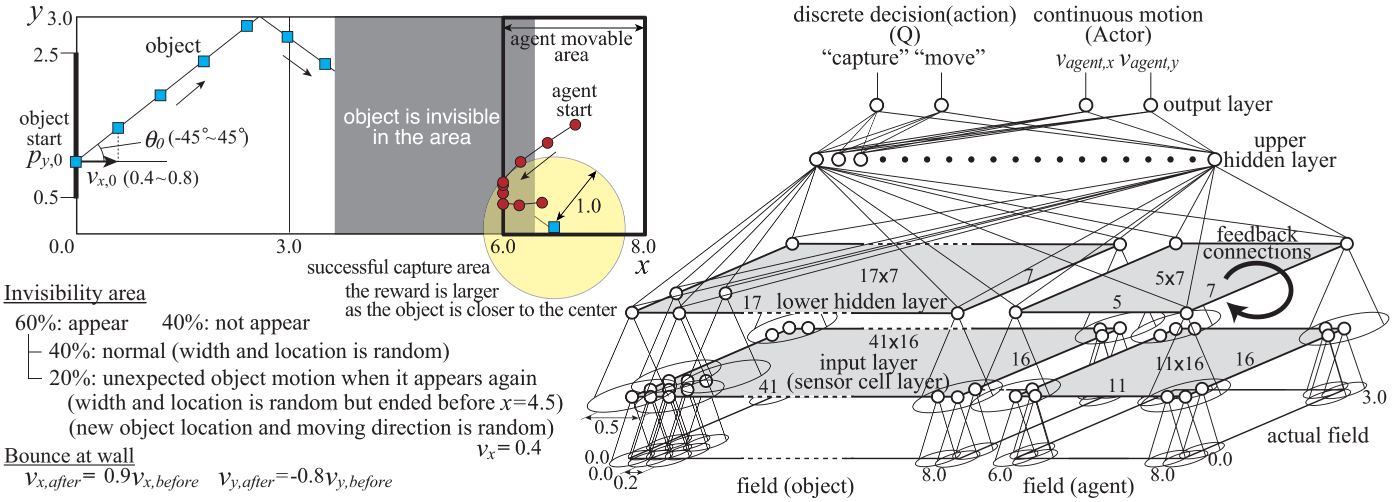
§ **Exploration**
Effective and deterministic exploration behavior for ambiguous or invisible goal considering past experience was learned and temporal abstraction was discussed[24, 25].

§ **Communication** (introduced in another paper[26])

Although each function has been examined in a very simple task, it is known that a variety of functions emerge based on extraction and memory of necessary information using an RNN. However, the emergence of "thinking" or "symbol processing" that needs multi-step state transition has not been observed yet.

3.3 Frame-free Function Emergence for Artificial General Intelligence

As mentioned before, through end-to-end RL from sensors to motors, entire process is learned and comprehensive function is acquired. For example, in the work in [22], an agent who has a visual sensor and an RNN as shown in Fig. 3 learned both motion and capture timing of a moving object that often becomes invisible randomly.



y
3.0
2.5
0.5
0.0
3.0
6.0
8.0
x
object
object start
$p_{y,0}$
θ_0 (-45°~45°)
$v_{x,0}$ (0.4~0.8)
object is invisible in the area
agent movable area
agent start
1.0
successful capture area
the reward is larger as the object is closer to the center
Invisibility area
60%: appear 40%: not appear
40%: normal (width and location is random)
20%: unexpected object motion when it appears again
(width and location is random but ended before x=4.5)
(new object location and moving direction is random)
v_x= 0.4
Bounce at wall
$v_{x,after}$= 0.9$v_{x,before}$ $v_{y,after}$=-0.8$v_{y,before}$
discrete decision(action)
(Q)
"capture" "move"
continuous motion
(Actor)
$v_{agent,x}$ $v_{agent,y}$
output layer
upper hidden layer
feedback connections
17x7
17
lower hidden layer
7
5x7
5
7
41x16
input layer
(sensor cell layer)
41
16
11x16
16
11
3.0
actual field
0.5
0.0
0.0
0.2
field (object)
8.0
6.0
field (agent)
8.0
0.0

Figure 3: Invisible moving object capture problem (prediction task). Invisibility area, object start location, initial object moving direction and velocity are decided randomly at each episode. The input of the RNN is 832 visual signals (656 for the object and 176 for the agent itself). By using Actor-Q, the agent can choose 'capture' or 'move', and when 'move' is chosen, the continuous motion is determined by the two actor outputs.(2013)[22]

In a general approach, the object motion is estimated from some frames of image using some given model, the future object location is predicted, the capture point and time are decided by some optimization method, a reference trajectory is derived from the capture point, and the motions are controlled to follow the trajectory.

Fig. 4 shows four examples after RL based on the reward given for object capture. The agent did not know in advance the way to predict the motion or even the fact that prediction is necessary to catch it. Nevertheless, it moved to the very front of its range of motion, waited the object, and when the object came to close, the agent moved backward with it and caught it. Though the object became invisible or visible again suddenly, the agent could behave appropriately. Since the moving direction of the object changed sometimes when it was invisible during learning, the agent learned to wait close to the center ($y = 1.5$) where it can react the unexpected object motion. As shown in case 4, when the object changed its direction unexpectedly, the agent could catch it though the timing is a bit later than the case of expected motion (case 3).

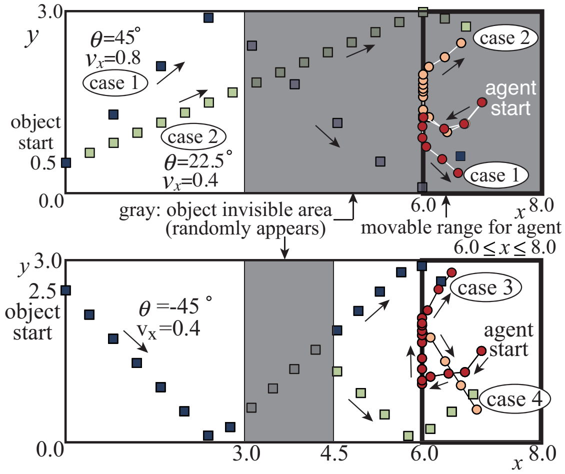


Figure 4: Sample agent behaviors after learning of a prediction task. (2013)[22]

References

[1] Minh, V., Kavukcuoglu, K., et al. (2013) Playing Atari with Deep Reinforcement Learning, *NIPS Deep Learning Workshop 2013*.

[2] Minh, V., Kavukcuoglu, K., Silver, D., et al. (2015) Human-level control through deep reinforcement learning *Nature*, **518**, 529–533.

[3] Silver, D., Huang, A., et al. (2016) Mastering the game of Go with deep neural networks and tree search, *Nature*, **529**, 484–489.

[4] Tesauro, G. (1992) Practical Issues in Temporal Difference Learning, *Machine Learning*, **8**, 257–277.

[5] Krizhevsky, A., Sutskever, I., et al. (2012) ImageNet Classification with Deep Convolutional Neural Networks *NIPS*, **25**, 1097–1105.

[6] Amodei, D., Anubhai, R., et al. (2015) DeepSpeech2: End-to-end Speech Recognition in English and Mandarin *arXiv*:1512.02595.

[7] Shibata, K. & Okabe, Y. (1997) Reinforcement Learning When Visual Sensory Signals are Directly Given as Inputs, *Proc. of ICNN(Int'l Conf. on Neural Networks)97*, **3**, 1716–1720.

[8] Shibata, K., Okabe, Y. & Ito, K. (1998) Direct-Vision-Based Reinforcement Learning in "Going to an Target" Task with an Obstacle and with a Variety of Target Sizes, *Proc. of NEURAP(Neural Networks and their Applications)'98*, 95-102.

[9] Shibata, K. (2011) Emergence of Intelligence through Reinforcement Learning with a Neural Network, *Advances in Reinforcement Learning*, Intech, 99–120.

[10] Shibata, K. & Iida, M. (2003) Acquisition of Box Pushing by Direct-Vision-Based Reinforcement Learning, *Proc. of SICE Annual Conf. 2003*, 1378–1383.

[11] Shibata, K. & Kawano, T. (2008) Learning of Action Generation from Raw Camera Images in a Real-World-like Environment by Simple Coupling of Reinforcement Learning and a Neural Network, *Adv. in Neuro-Information Processing, LNCS*, **5506**, 755–762.

[12] Shibata, K. & Kawano, T. (2009) Acquisition of Flexible Image Recognition by Coupling of Reinforcement Learning and a Neural Network, *SICE J. of Control, Measurement, and System Integration*, **2**(2), 122–129.

[13] Shibata, K. & Kurizaki, S. (2012) Emergence of Color Constancy Illusion through Reinforcement Learning with a Neural Network, *Proc. of ICDL-EpiRob(Int'l Conf. on Developmental Learning & Epigenetic Robotics)2012*, PID2562951.

[14] Shibata, K., Nishino, T. & Okabe, Y. (1995) Active Perception Based on Reinforcement Learning, *Proc. of WCNN'95*, **II**, 170–173.

[15] Shibata, K. & Ito, K. (2002) Effect of Force Load in Hand Reaching Movement Acquired by Reinforcement Learning, *Proc. of ICONIP(Int'l Conf. on Neural Information Processing)2002*, 1444–1448.

[16] Shibata, K. & Ito, K. (2003) Hidden Representation after Reinforcement Learning of Hand Reaching Movement with Variable Link Length, *Proc. of IJCNN (Int'l Joint Conf. on Neural Networks) 2003*, 2619–2624.

[17] Shibata, K. (2006) Spatial Abstraction and Knowledge Transfer in Reinforcement Learning Using a Multi-Layer Neural Network, *Proc. of ICDL (5th Int'l Conf. on Development and Learning) 2006*, 36 (CD-ROM).

[18] Utsunomiya, H. & Shibata, K. (2008) Contextual Behavior and Internal Representations Acquired by Reinforcement Learning with a Recurrent Neural Network ..., Adv. in Neuro-Information Processing, Lecture Notes in Comp. Sci., Proc. of ICONIP '08,*5507*, 970–978

[19] Shibata, K. & Utsunomiya, H. (2011) Discovery of Pattern Meaning from Delayed Rewards by Reinforcement Learning with a Recurrent Neural Network, *Proc. of IJCNN (Int'l Joint Conf. on Neural Networks) 2011*, 1445–1452.

[20] Faudzi, A. A. M. & Shibata, K. (2014) Acquisition of Context-Based Active Word Recognition by Q-Learning Using a Recurrent Neural Network, *Robot Intelligent Technology and Applications*, **2**, 191–200.

[21] Shibata, K. & Sugisaka, M. (2004) Dynamics of a Recurrent Neural Network Acquired through Learning of a Context-based Attention Task, *Artificial Life and Robotics*, **7** (4), 145–150.

[22] Shibata, K. & Goto, K. (2013) Emergence of Flexible Prediction-Based Discrete Decision Making and Continuous Motion Generation through Actor-Q-Learning, *Proc. of ICDL-Epirob (Int'l Conf. on Developmental Learning & Epigenetic Robotics) 2013*, ID 15.

[23] Ishii S., Shidara, M. & Shibata, K. (2006) A model to explain the emergence of reward expectancy neurons using reinforcement learning and neural network, *Neurocomputing*, **69**, 1327–1331.

[24] Shibata, K. (2006) Learning of Deterministic Exploration and Temporal Abstraction in Rein..., *Proc. of SICE-ICCAS 2006*, 4569–4574.

[25] Goto, K. & Shibata, K. (2010) Acquisition of Deterministic Exploration and Purposive Memory through Reinforcement Learning with a Recurrent Neural Network, *Proc. of SICE Annual Conf. 2010*, FB03-1.pdf

[26] Shibata, K.(2017) Communications that Emerge through Reinforcement Learning Using a (Recurrent) Neural Network, RLDM2017

Unsupervised Basis Function Adaptation for Reinforcement Learning

Edward W. Barker
School of Mathematics and Statistics
University of Melbourne
Melbourne, Australia
`ebarker@student.unimelb.edu.au`

Charl J. Ras
School of Mathematics and Statistics
University of Melbourne
Melbourne, Australia
`cjras@unimelb.edu.au`

Abstract

When using reinforcement learning (RL) algorithms to evaluate a policy it is common, given a large state space, to introduce some form of approximation architecture for the value function (VF). The exact form of this architecture can have a significant effect on the accuracy of the VF estimate, however, and determining a suitable approximation architecture can often be a highly complex task. Consequently there is a large amount of interest in the potential for allowing RL algorithms to adaptively generate approximation architectures.

We investigate a method of adapting approximation architectures which uses feedback regarding the frequency with which an agent has visited certain states to guide which areas of the state space to approximate with greater detail. This method is "unsupervised" in the sense that it makes no direct reference to reward or the VF estimate. We introduce an algorithm based upon this idea which adapts a state aggregation approximation architecture on-line.

A common method of scoring a VF estimate is to weight the squared Bellman error of each state-action by the probability of that state-action occurring. Adopting this scoring method, and assuming S states, we demonstrate theoretically that — provided (1) the number of cells X in the state aggregation architecture is of order $\sqrt{S} \ln S \log_2 S$ or greater, (2) the policy and transition function are close to deterministic, and (3) the prior for the transition function is uniformly distributed — our algorithm, used in conjunction with a suitable RL algorithm, can guarantee a score which is arbitrarily close to zero as S becomes large. It is able to do this despite having only $O(X \log_2 S)$ space complexity and negligible time complexity. The results take advantage of certain properties of the stationary distributions of Markov chains.

Keywords: reinforcement learning, unsupervised learning, basis function adaptation, state aggregation

1 Introduction

When using traditional reinforcement learning (RL) algorithms (such as Q-learning or SARSA) to evaluate policies in environments with large state or action spaces, it is common to introduce some form of architecture with which to approximate the value function (VF), for example a parametrised set of functions. This approximation architecture allows algorithms to deal with problems which would otherwise be computationally intractable. One issue when introducing VF approximation, however, is that the accuracy of the algorithm's VF estimate is highly dependent upon the exact form of the architecture chosen.

Accordingly, a number of authors have explored the possibility of allowing the approximation architecture to be *learned* by the agent, rather than pre-set manually by the designer (see [1] for an overview). It is common to assume that the approximation architecture being adapted is linear (so that the value function is represented as a weighted sum of basis functions) in which case such methods are known as *basis function adaptation*.

A simple and perhaps, as yet, under-explored method of basis function adaptation involves using an estimate of the frequency with which an agent has visited certain states to determine which states to more accurately represent. Such methods are *unsupervised* in the sense that no direct reference to the reward or to any estimate of the value function is made. The concept of using visit frequencies in an unsupervised manner is not completely new [3][5] however it remains relatively unexplored when compared to methods based on direct estimation of VF error [2][6][7][8].

However it is possible that such methods can offer some unique advantages. In particular: (i) estimates of visit frequencies are very cheap to calculate, (ii) accurate estimates of visit frequencies can be generated with a relatively small number of samples, and, perhaps most importantly, (iii) in many cases visit frequencies contain a lot of the most important information regarding where accuracy is required in the VF estimate. Our aim here is to further explore and quantify, where possible, these advantages.

In the next section we outline the details of an algorithm (PASA, short for "Probabilistic Adaptive State Aggregation") which performs unsupervised basis function adaptation based on state aggregation. This algorithm will form the basis upon which we develop theoretical results in Section 3. Our main theoretical result, which relates to point (iii) above, demonstrates that circumstances exist where PASA can ensure arbitrarily low VF error (given an appropriate scoring function) and will therefore perform just as well as many alternative more complex methods, despite having a much lower associated cost. The result is important because it highlights the potential of unsupervised methods, and also because it provides insight into the circumstances where such methods are likely to be most effective.

2 The PASA algorithm

2.1 Formal setting

The agent interacts with an environment over a sequence of iterations $t \in \mathbb{N}$. For each t it will be in a particular state s_i ($1 \leq i \leq S$) and will take a particular action a_j ($1 \leq j \leq A$) according to a policy π (we denote as $\pi(a_j|s_i)$ the probability the agent takes action a_j in state s_i). Transition and reward functions are denoted as P and R respectively (and are unknown, however we assume we are given a prior distribution for both). Hence we use $P(s_{i'}|s_i, a_j)$ to denote the probability the agent will transition to state $s_{i'}$ given it takes action a_j in state s_i. We assume the reward function (which maps each state-action pair to a real number) is deterministic and bounded: $|R(s_i, a_j)| < \infty$ for all (i, j). We are considering the problem of policy evaluation, so we will always assume that the agent's policy π is fixed. A *state aggregation* approximation architecture we will define as a mapping F from each state s_i to a *cell* x_k ($1 \leq k \leq X$), where typically $X \ll S$ [4]. Given a state aggregation approximation architecture, an RL algorithm will maintain an estimate $\hat{Q}_\theta$ of the true value function Q^π [1], where θ specifies a weight associated with each cell-action pair. Provided for any particular mapping F the value θ converges, then each mapping F has a corresponding VF estimate for the policy π.

Many different methods can be used to score a VF estimate (i.e. measure its accuracy). A common score used is the squared error in the VF estimate for each state-action, weighted by the probability of visiting each state. This is called the *mean squared error* (MSE). When the true VF Q^π is unknown we can use T, the *Bellman operator*, to obtain an approximation of the MSE.[1] This approximation we denote as L. Hence:

$$\begin{aligned}\text{MSE} := \sum_{i=1}^{S} \psi_i \sum_{j=1}^{A} \Big(Q^\pi(s_i, a_j) - \hat{Q}_\theta(s_i, a_j) \Big)^2 \approx L &:= \sum_{i=1}^{S} \psi_i \sum_{j=1}^{A} \Big(T\hat{Q}_\theta(s_i, a_j) - \hat{Q}_\theta(s_i, a_j) \Big)^2 \\ &= \sum_{i=1}^{S} \psi_i \sum_{j=1}^{A} \left(R(s_i, a_j) + \gamma \sum_{i'=1}^{S} \sum_{j'=1}^{A} P(s_{i'}|s_i, a_j) \pi(a_{j'}|s_{i'}) \hat{Q}_\theta(s_{i'}, a_{j'}) - \hat{Q}_\theta(s_i, a_j) \right)^2 \qquad (1)\end{aligned}$$

---

[1] See, for example, score functions proposed by Menache et al [3] and Di Castro et al [7].

This is where $\gamma \in [0,1)$ is a discount factor and where ψ is a vector of the probability of each state given the stationary distribution associated with π (given some fixed policy π, the transition matrix, obtained from π and P, has a corresponding stationary distribution). We assume that our task in developing an adaptive architecture is to design an algorithm which will adapt F so that L is minimised. The PASA algorithm, which we now outline, adapts the mapping F.

2.2 Details of PASA algorithm

The key idea underlying PASA is that it will generate on-line a state aggregation architecture such that frequently visited states occupy "small" cells and infrequently visited states occupy "large" cells. Designing an algorithm to achieve this would be a relatively easy task if we could store an estimate of ψ_i for every state, however by assumption S is too large for this to be possible. Hence we must introduce into our algorithm some steps to ensure its memory requirements are not excessive. In particular, PASA estimates *cell* visit frequencies (via an indirect, iterative method) as opposed to *state* visit frequencies, and this information is translated on-line into a suitable state aggregation architecture.

The details are as follows: PASA will store a vector ρ of integers of dimension $X - B$, where $B < X$. Suppose we start with a partition of the state space into B cells, indexed from 1 to B, each of which is approximately the same size. Using ρ we can now define a new partition by splitting (as evenly as possible) the ρ_1th cell in this partition. We leave one half of the ρ_1th cell with the index ρ_1 and give the other half the index $B+1$ (all other indices stay the same). Taking this new partition (consisting of $B+1$ cells) we can create a further partition by splitting the ρ_2th cell. Continuing in this fashion we will end up with a partition containing X cells (which gives us the mapping F). We need some additional mechanisms to allow us to update ρ. Denote as $\mathcal{X}_{i,j}$ the set of states in the ith cell of the jth partition (so $0 \leq j \leq X - B$ and $1 \leq i \leq B + j$). PASA will store a vector $\bar{u}$ of real values of dimension X (which indirectly records the approximate frequency with which certain cells have been visited by the agent). We define a new vector $\bar{x}$ of dimension X:

$$\bar{x}_i(t) = \begin{cases} I_{\{s(t)\in\mathcal{X}_{i,0}\}} & \text{if } 1 \leq i \leq B \\ I_{\{s(t)\in\mathcal{X}_{i,i-B}\}} & \text{if } B < i \leq X \end{cases} \tag{2}$$

where I is the indicator function for a logical statement (such that $I_A = 1$ if A is true). The resulting mapping from each state to a vector $\bar{x}$ we denote as $\bar{F}$. We then update $\bar{u}$ in each iteration as follows (i.e. via a stochastic approximation):

$$\bar{u}_i(t+1) = \bar{u}_i(t) + \eta\left(\bar{x}_i(t) - \bar{u}_i(t)\right) \tag{3}$$

This is where $\eta \in (0,1]$ is a constant step size parameter. To update ρ, at certain intervals $\nu \in \mathbb{N}$ the PASA algorithm performs a sequence of $X - B$ operations. A temporary copy of $\bar{u}$ is made, which we call u. We also store an X dimensional boolean vector Σ and set each entry to zero at the start of the sequence (this keeps track of whether a particular cell has only one state, as we don't want singleton cells to be split). At each stage $k \geq 1$ of the sequence we update ρ, u and Σ, in order, as follows (for ρ, if multiple indices satisfy the $\arg\max$ function, we take the lowest index):

$$\begin{aligned} &\rho_k = \begin{cases} \arg\max_i\{u_i : i \leq B+k-1, \Sigma_i = 0\} & \text{if } (1-\Sigma_{\rho_k})u_{\rho_k} < \max\{u_i : i \leq B+k-1, \Sigma_i = 0\} - \vartheta \\ \rho_k & \text{otherwise} \end{cases} \\ &u_{\rho_k} \leftarrow u_{\rho_k} - u_{B+k} \qquad \Sigma_i = I_{\{|\mathcal{X}_{i,k}|\leq 1\}} \text{ for } 1 \leq i \leq B+k \end{aligned} \tag{4}$$

where $\vartheta > 0$ is a constant designed to ensure that a (typically small) threshold must be exceeded before ρ is adjusted. The idea behind each step in the sequence is that the non-singleton cell $\mathcal{X}_{i,k}$ with the highest value u_i (an estimate of visit frequency which is recalculated at each step) will be split. The overall PASA process is outlined in Algorithm 1.[2]

Algorithm 1 The PASA algorithm. Called at each iteration t. Assumes $\bar{u}$, $\bar{F}$, F and ρ are stored. Return is void.

```
function PASA($t$,$s$,$\eta$,$\vartheta$,$\nu$)
    $\bar{x} \leftarrow \bar{F}(s)$
    $\bar{u} \leftarrow \bar{u} + \eta(\bar{x} - \bar{u})$
    if $t \bmod \nu = 0$ then
        $u \leftarrow \bar{u}$
        for $k \in \{1, \ldots, X\}$ do
            $\Sigma_k \leftarrow 0$
        end for
        for $k \in \{1, \ldots, X - B\}$ do
            $u_{\max} \leftarrow \max\{u_i : \Sigma_i = 0\}$
            $i_{\max} \leftarrow \min\{i : u_i = u_{\max}, \Sigma_i = 0\}$
            if $u_{i_{\max}} - \vartheta > (1 - \Sigma_{\rho_k})u_{\rho_k}$ then
                $\rho_k \leftarrow i_{\max}$
            end if
            $u_{\rho_k} \leftarrow u_{\rho_k} - u_{B+k}$
            $(\bar{F}, \Sigma) \leftarrow$ SPLIT$(k, \rho, \bar{F})$
        end for
        $F \leftarrow$ CONVERT$(\bar{F})$
    end if
end function
```

---

[2] Note that the algorithm calls a procedure to SPLIT cells. This procedure simply updates $\bar{F}$ and Σ given the latest value of ρ (how the cell is split is arbitrary, subject to our assumption that the split results in two cells of roughly equal size). It also calls a CONVERT procedure, which converts the mapping $\bar{F}$ to a mapping F, a straightforward operation based on the definitions of $\bar{F}$ and F.

2.3 Some basic properties of PASA

It is possible to demonstrate (we omit the details) that, given fixed π, PASA will converge in the sense that, when suitable assumptions are applied to the parameter η (in particular that it is sufficiently low), ρ will eventually remain the same value over arbitrarily long sequences of iterations with arbitrarily high probability.

PASA requires only a modest increase in computational resources compared to fixed state aggregation. In relation to time complexity, $\bar{u}$ can be updated in parallel with the RL algorithm's update of θ (and the update of $\bar{u}$ would not be expected to have any greater time complexity than the update to θ if using a standard RL algorithm such as SARSA), whilst ρ can be updated at large intervals ν (and this update can also be run in parallel). Hence, PASA involves no material increase in time complexity. PASA *does* involve additional space complexity with respect to storing the vector $\bar{u}$: we must store X real values. If we also store F and $\bar{F}$ (as well as u temporarily) the overall space complexity becomes $O(X \log_2 S)$. The RL component has space complexity $O(XA)$ (reflecting the $X \times A$ cell-action pairs), so that the introduction of PASA as a pre-processing algorithm will not impact the overall space complexity at all if $A > \log_2 S$. (Note also that the space complexity of PASA is independent of A.) Regarding sampling efficiency, since methods based on explicitly estimating the Bellman error (or MSE) require a VF estimate (generated by the RL algorithm), as well as information about reward for all actions in the action space, we can expect PASA (and other unsupervised methods) to require comparatively less sampling to generate the estimates it requires to update the approximation architecture.

3 Result regarding Bellman error for policy evaluation

We now set out our main result. The key idea is that, in many important circumstances, which are reflective of real world problems, when following a fixed policy (even when this is generated randomly) an agent will have a tendency to spend nearly all of its time in only a small subset of the state space. We can use this property to our advantage. It means that by focussing on this small area (which is what PASA does) we can eliminate most of the terms which significantly contribute to L. The trick will be to quantify this tendency. We must make the following assumptions: (1) P is "close to" deterministic, i.e. P, interpreted as an operator with three arguments, can expressed as follows: $P = (1-\delta)P_1 + \delta P_2$, where P_1 is a deterministic transition function and P_2 is an arbitrary transition function, and where δ is small (what constitutes "small" will be made clearer below), (2) P has a uniform *prior* distribution, in the sense that, according to our prior distribution for P, the random vector $P(\cdot|s_i, a_j)$ is independently distributed for all (i,j) and each random variable $P(s_{i'}|s_i, a_j)$ is identically distributed for all (i, j, i'), and (3) π is also "close to" deterministic (i.e. the probability of *not* taking the most probable action is no greater than δ for each state).

We can make the following observation. If π and P are deterministic, and we pick a starting state s_1, then the agent will create a path through the state space and will eventually revisit a previously visited state, and will then enter a cycle. Call the set of states in this cycle $\mathcal{C}_1$ and denote as C_1 the number of states in the cycle. If we now place the agent in a state s_2 (arbitrarily chosen) it will either create a new cycle or it will terminate on the path or cycle created from s_1. Call $\mathcal{C}_2$ the states in the second cycle (and C_2 the number of states in the cycle, noting that $C_2 = 0$ is possible). If we continue in this manner we will have S sets $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_S\}$. Call $\mathcal{C}$ the union of these sets and denote as C the number of states in $\mathcal{C}$. We denote as T_i the event that the ith path created in such a manner terminates on itself, and note that, if this does not occur, then $C_i = 0$. If (2) holds then $\mathrm{E}(C_1) = \sqrt{\pi S/8} + O(1)$ and $\mathrm{Var}(C_1) = (32 - 8\pi)S/24 + O(\sqrt{S})$.[3] Supposing that π and P are no longer deterministic then, if (1) and (3) hold, we can set δ sufficiently low so that the agent will spend an arbitrarily large proportion of its time in $\mathcal{C}$. If (2) holds we also have the following:

Lemma 1. $\mathrm{E}(C) < \mathrm{E}(C_1)(\ln S + 1)$ and $\mathrm{Var}(C) \leq O(S \ln S)$.

Proof. We will have:

$$\mathrm{E}(C) = \sum_{i=1}^{S} \mathrm{E}(C_i) = \sum_{i=1}^{S} \Pr(T_i) \sum_{j=1}^{S} j \Pr(C_i = j | T_i) \leq \sum_{i=1}^{S} \frac{1}{i} \sum_{j=1}^{S} j \Pr(C_1 = j) < \mathrm{E}(C_1)(\ln S + 1) \tag{5}$$

And for the variance:

$$\begin{aligned}\mathrm{Var}(C) &= \sum_{i=1}^{S} \mathrm{Var}(C_i) + 2\sum_{i=2}^{S}\sum_{j=1}^{i-1} \mathrm{Cov}(C_i C_j) \leq \sum_{i=1}^{S} \mathrm{Var}(C_i) \leq \sum_{i=1}^{S} \mathrm{E}(C_i^2) = \sum_{i=1}^{S} \Pr(T_i) \sum_{j=1}^{S} j^2 \Pr(C_i = j|T_i) \\ &\leq \sum_{i=1}^{S} \frac{1}{i} \sum_{j=1}^{S} j^2 \Pr(C_1 = j) < \mathrm{E}(C_1^2)(\ln S + 1) = \left(\mathrm{Var}(C_1) + \mathrm{E}(C_1)^2\right)(\ln S + 1)\end{aligned} \tag{6}$$

---

[3] The expectation is over the prior distribution for P. The result can be shown by adapting the solution to the "birthday problem". For a description of the problem and a formal proof see, for example, page 114 of Flajolet and Sedgewick [9]. The variance can be derived from first principles using similar techniques to those used for the mean in the birthday problem.

where we have used the fact that the covariance term must be negative for any pair of lengths C_i and C_j, since if C_i is greater than its mean the expected length of C_j must decrease, and vice versa. □

Theorem 2. For all $\epsilon_1 > 0$ and $\epsilon_2 > 0$, there is sufficiently large S and sufficiently small δ such that PASA in conjunction with a suitable RL algorithm will — provided $X \geq K\sqrt{S}\ln S \log_2 S$ for some $K > \sqrt{\pi/8}$ — generate, with probability no less than $1 - \epsilon_1$, a VF estimate with $L \leq \epsilon_2$.

Proof. Using Chebyshev's inequality, and Lemma 1, we can choose S sufficiently high so that $C > K\sqrt{S}\ln S$ with probability no greater than ϵ_1. Since R is bounded and $\gamma < 1$ then for any $\epsilon_2 > 0$, F and S we can also choose δ so that L summed only over states not in $\mathcal{C}$ is no greater than ϵ_2. We choose δ so that this is satisfied, but also so that $\psi_i > \sum_{i' : s_{i'} \notin \mathcal{C}} \psi_{i'}$ for all elements of $\{\psi_i : s_i \in \mathcal{C}\}$.[4] Now provided that $C \log_2 S \leq X$ then each state in $\mathcal{C}$ will eventually be in its own cell. The RL algorithm will have no error for each such state so therefore L will be no greater than ϵ_2. □

The bound on X provided represents a significant reduction in complexity when S starts to take on a size comparable to many real world problems (and could make the difference between a problem being tractable and intractable). It's also possible that the bound on X in Theorem 2 can be improved upon. Conditions (1) and (3) are commonly encountered in practice, in particular (3) which can be taken to reflect a "greedy" policy. Condition (2) can be interpreted as the transition function being "completely unknown". It is perhaps the most restrictive of the three conditions. However some scope exists (which we do not explore here) to use similar techniques to extend the results to more general priors. In particular, provided the prior distribution for P is such that the expected size of $\mathcal{C}$ is small compared to S, then similar concepts to those used in the proof of Theorem 2 will apply. Note finally that the result can be extended to exact MSE if MSE is redefined so that it is also weighted by π.

4 Discussion

The key message from our discussion is that there are important circumstances where unsupervised methods can be very effective in creating an approximation architecture. However, given their simplicity, they can at the same time avoid the cost (both in terms of computational complexity, and sampling required) associated with more complex adaptation methods. In the setting of policy *improvement* these advantages have the potential to be particularly important, especially when dealing with large state spaces. The theoretical implications of extending unsupervised methods to policy iteration remains an interesting question for further research.

Given that our discussion is largely theoretical, important questions remain in relation to how well unsupervised methods might perform in practice. Some initial experimentation suggests that the PASA algorithm can have a significant impact on RL algorithm performance in a practical setting in relation to both policy evaluation and policy improvement. In a practical setting, moreover, evidence suggests that many of the more extreme assumptions which were necessary in order to obtain rigorous theoretical results (for example very low δ) can potentially be relaxed without preventing PASA from improving performance, further highlighting some of the unique advantages of such methods.

References

[1] L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators.* Taylor & Francis CRC Press, 2009.

[2] R. Munos and A. Moore. *Variable resolution discretization in optimal control.* Machine Learning, 49(2-3):291-323, 2002.

[3] I. Menache, S. Mannor, and N. Shimkin. *Basis function adaptation in temporal difference reinforcement learning.* Annals of Operations Research, 134(1):215-238, 2005.

[4] S. Whiteson, M. Taylor and P. Stone. *Adaptive tile coding for value function approximation.* University of Texas at Austin, Technical Report, 2007.

[5] A. Bernstein and N. Shimkin. *Adaptive-resolution reinforcement learning with efficient exploration in deterministic domains.* Machine Learning, 81(3):359-397, 2010.

[6] D. Bertsekas and H. Yu. *Convergence results for some temporal difference methods based on least squares.* IEEE Transactions on Automatic Control, 54(7):1515-1531, 2009.

[7] D. Di Castro and S. Mannor. *Adaptive bases for reinforcement learning.* Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 312-327, 2010.

[8] S. Mahadevan, S. Giguere, and N. Jacek. *Basis adaptation for sparse nonlinear reinforcement learning.* AAAI, 2013.

[9] P. Flajolet and R. Sedgewick. *Analytic Combinatorics.* Cambridge University Press, 2009.

---

[4] Each such ψ_i will be bounded from below for all $\delta > 0$. This can be verified by more closely examining the geometric distributions which govern the "jumping" between distinct cycles.

Deciding to Specialize and Respecialize a Value Function for Relational Reinforcement Learning

Mitchell Keith Bloch
University of Michigan
Ann Arbor, MI, USA
`bazald@umich.edu`

John Edwin Laird
University of Michigan
Ann Arbor, MI, USA
`laird@umich.edu`

Abstract

We investigate the matter of feature selection in the context of relational reinforcement learning. We had previously hypothesized that it is more efficient to specialize a value function quickly, making specializations that are potentially suboptimal as a result, and to later modify that value function in the event that the agent gets it "wrong." Here we introduce agents with the ability to adjust their generalization through respecialization criteria. These agents continuously reevaluate the feature selection problem to see if they should change how they have structured their value functions as they gain more experience. We present performance and computational cost data for these agents and demonstrate that they can do better than the agents with no ability to revisit the feature selection problem.

Keywords: relational reinforcement learning, feature selection

1 Introduction

Bloch and Laird [2015] demonstrated the computational and learning efficiency of the Rete algorithm in the context of relational reinforcement learning (RRL) and explored the quality of learning under several feature selection criteria. This prior work involved feature selection criteria that allowed agents to decide when and how to specialize the value function. It provided no mechanisms for evaluating the quality of these value function specializations after the fact, so any decisions that were made were then final.

They hypothesized that it is more efficient to specialize the value function quickly, making specializations that are potentially suboptimal as a result, and to later modify the value function in the event that the agent gets it "wrong." We now provide our agents with criteria to allow them to respecialize the value function using different features at a given level of generality. We present these results in section 3.

2 Background

We're concerned with the version of the reinforcement learning problem defined as a minimization of cumulative regret over the course of learning. The execution cycle for one of our agents is similar to many reinforcement learning agents learning with temporal difference methods. The agent chooses an exploratory or greedy action, gets a reward from the resulting state transition, and then learns from the complete $<s, a, r, s', a'>$ tuple. Our agents learn online using the modern temporal difference method, GQ(λ) [Maei and Sutton, 2010]. GQ(λ) is a good choice because it supports online, incremental learning and provides convergence guarantees when using linear function approximation. Any other algorithm with the same properties would suffice.

2.1 Relational Blocks World with Goal Configuration

There are many formulations of the classic Blocks World domain. Blocks World typically presents a configuration of blocks, each with a unique label. The agent is tasked with moving one block at a time from the top of a stack of blocks to either the top of a different stack of blocks or to the table. The agent completes the task when the blocks' configuration matches the goal configuration.

The Blocks World task that interests us exposes the goal configuration to the agent, as depicted in figure 1. Distractor features are included to ensure that the best answer to the feature selection problem is not simply to include all features of the environment. Agents using a standard, fixed propositional representation of this formulation of the Blocks World task cannot learn a near optimal strategy for solving new goal configurations because they must ultimately capture some details of the task specific to a particular goal configuration in order to function at all. Agents using relational representations can capture connections between different features of the environment in a more general way, allowing their policies to generalize between different initial conditions, goal configurations, and even numbers of blocks. See section 2.2 for details.

The relational features available to our agents are: whether the source stack of blocks matches a goal stack or not; whether the destination stack matches a goal stack or not; whether the top block of the source stack belongs on top of the destination stack or not (a top block being considered to match the top of the destination stack if and only if the destination stack matches a goal stack and the additional block will allow it to continue to match); and whether the destination (stack) is the table. The distrator features available to our agents include the names of blocks; the heights both of the source stack and of the destination stack; a randomized brightness or luminosity value for the blocks on top of both the source stack and the destination stack; and a related value as to whether that block is considered to be glowing or not.

2.2 Relational Reinforcement Learning

Relational reinforcement learning is an approach to support generalization from experience [Tadepalli *et al.*, 2004; Džeroski *et al.*, 2001]. Given relations and conjunctions thereof, it is possible to condense both the state-space and action-space for certain tasks.

For a minimal one-feature example, if the agent is concerned that a destination stack of blocks in the environment matches a stack of blocks in the goal configuration, that feature can expressed relationally as: `(`$dest\text{-}stack$`, matches, `$goal\text{-}stack$`)`. Given this formulation, the agent can learn about all actions in which that match is present, regardless of the details of the specific labels of any of the blocks in either the $dest\text{-}stack$ or the $goal\text{-}stack$.

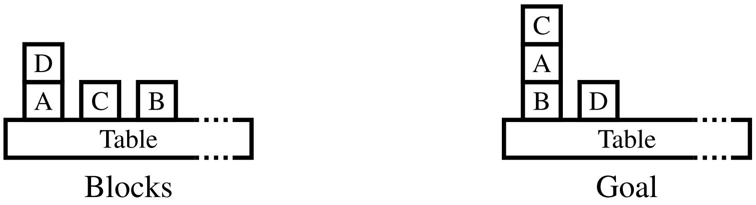


Figure 1: Visual representation of both the blocks the agent sees in the world and the goal configuration provided to the agent.

An agent could instead consider a feature expressing whether a block belongs on top of a destination stack of blocks in the environment: `(`$block$`, matches-top,` $dest{-}stack$`)`. This relation again expresses a general concept and does not capture the details of the specific labels of either the block, $block$, nor of any of the blocks in the $goal{-}stack$.

If both features are important, an agent ought to consider the compound features derived from the conjunction of both `(`$dest{-}stack$`, matches,` $goal{-}stack$`)` and `(`$block$`, matches-top,` $dest{-}stack$`)`. This is where the real power of relational reinforcement learning comes in. Not only are both features general due to the use of variables in the place of what would otherwise be specific details of the environment, but the value function ensures that identical variables in both features match. Therefore, even though specifics of the environment are not captured by the value function, it can guarantee that both conditions refer to the same $dest\text{-}stack$ in this compound feature.

In the above example, given a conjunction of two Boolean features, four different possibilities will result. As features are combined, a combinatorial explosion can occur. However, as these features do not always depend on the size of the state-space and action-space, they can be particularly effective for scenarios with arbitrarily large state-spaces. As relational reinforcement learning allows an agent to reason about objects more abstractly, it additionally allows the agent to generalize from its experience to solve problems from outside its training set. If we were to propositionalize the set of features to allow it to learn this particular task, it would need complex conjunctions of many features to be filled out with every possible label in the environment, resulting in a similar combinatorial explosion but with none of the capability for generalization.

2.3 Value Functions

As in previous work, we use a value function using both linear function approximation and a dynamically specialized tile coding [Bloch and Laird, 2013; Bloch and Laird, 2015].

Linear function approximation contributes to the capability of our agents to generalize from past experience. Given n basis functions, $\phi_1, \ldots, \phi_n$, the Q-function is estimated by $Q(s,a) = \sum_{i=1}^{n} \phi_i(s,a) w_i$, where $\phi_i(s,a)$ is commonly 1 or 0 to indicate whether the feature is currently active for a given state and action, or not, and where the weight, w_i, represents the value contribution of the feature for the action under consideration, or a partial Q-value. That each feature applies to many states provides the agent with the ability to learn about states which the agent has yet to visit.

Starting with a coarse tile coding and breaking the tiles into smaller, more specific tiles over time also contributes to the capability of our agents to generalize from past experience. As the agent specializes its value function, it learns a more specialized policy, but without the full cost associated with starting with a maximally specialized value function from the beginning. We combine this with linear function approximation by keeping coarser, more general tiles in the system and allowing general learning to continue even as the smaller, more specialized tiles are added [Bloch and Laird, 2013].

3 Specializing and Respecializing

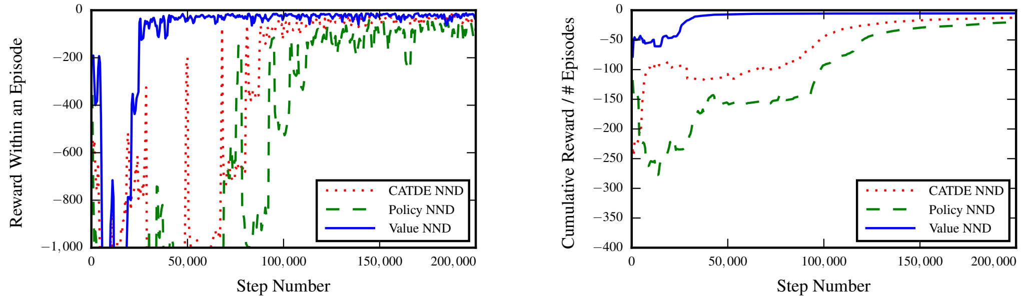
Our agents are trained on-policy with a discount rate of 0.9 and an eligibility trace decay rate of 0.3. We experimented with three value function specialization (feature selection) criteria: a cumulative absolute temporal difference error (CATDE) criterion, choosing features with the largest experienced cumulative temporal difference error; a policy criterion, choosing features that are most likely to result in changes to the policy based on estimated changes to the resulting value function; and a value criterion, choosing features that will result in the largest changes to the value estimates provided by the value function. Our policy and value criteria are directly inspired by those implemented by Whiteson *et al.* [2007]. To stress our learning algorithms, we provide them with two versions of several of the relational features: ones that provide correct information until step $5,000$ and then incorrect afterward; and ones that do the opposite. Initial block configurations and goal configurations of 4 blocks are randomly generated for each episode. Figure 2 depicts performance averages for the agents learning over 30 independent runs.

Bloch and Laird's [2015] hypothesis was that it is more efficient to specialize the value function quickly, making specializations that are potentially suboptimal as a result, and to later modify that value function in the event that the agent gets it "wrong." We expected this hypothesis to hold for all three criteria.

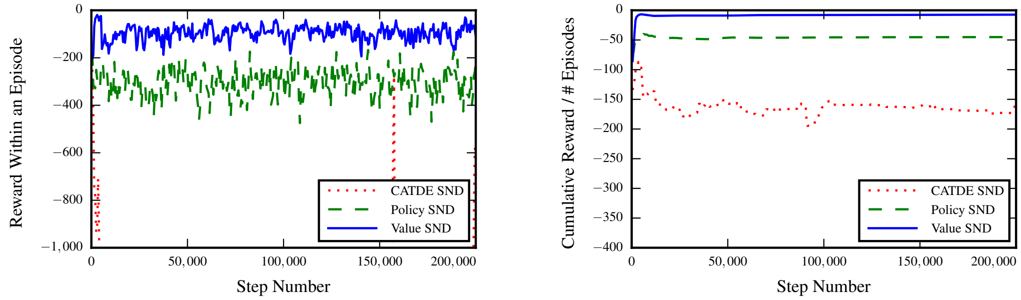
Figure 2a depicts the performance of agents that can only specialize the value function but are not allowed to change their minds about specializations they have made. It is evident that given that thrashing is impossible, the agents are eventually capable of learning good policies in the limit regardless of the choice of specialization criterion.

The first modification we make to our agents is to provide them with the ability to rerun their specialization criteria at each level of generalization of the value function at each time step. This allows our agents to compare metadata, policy decisions, and value estimates between the actual value function and versions of the value function in which each other specialization had been made instead, restricted to a one-step lookahead. These agents can then respecialize over a different set of features if it appears to be valuable to do so. Figure 2b depicts the performance of these agents. As these agents respecialize the value function with no mechanism in place to prevent thrashing, the agents using the policy and value criteria thrash to the extent that the value function stays very small. The agent using CATDE is comparably stable, but its computational cost is quite high as a result. It's worth noting that the agents using the value criterion still manage to do quite well on average despite significant thrashing and get computational savings that our later efforts do not come close to matching. This result is highlighted in pink in figure 3.

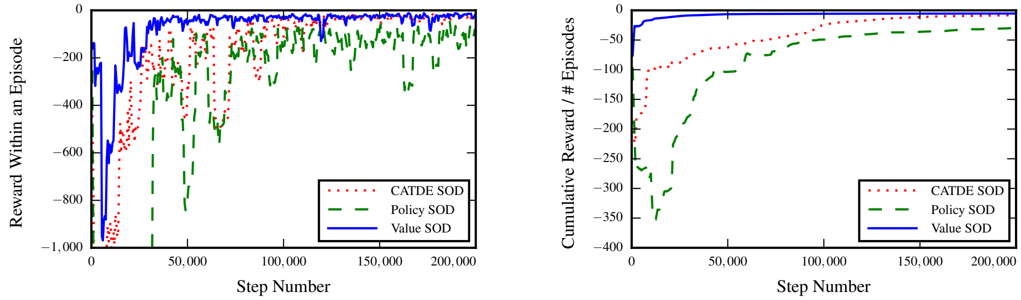
Our first attempt at a second modification of our agents is to implement a *blacklist* mechanism. This naive approach disallows an agent from choosing the same specialization twice at a given level of generalization (unless the value function is respecialized at an even more general level). The figure depicting the performance of the agents that additionally have this *blacklist* mechanism in place have been omitted because the performance is orders of magnitude worse than that of the agents with no respecialization. Thrashing likely results from the scenario in which a small number of choices are very good. Therefore, it is intuitive to expect blacklisting



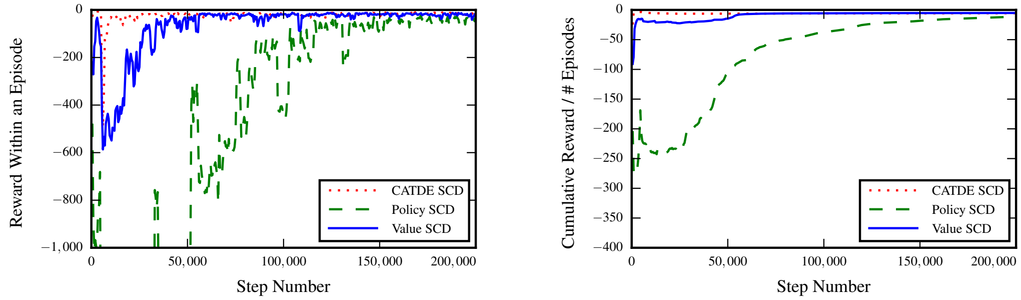
(a) No Despecialization



(b) Despecialization



(c) Despecialization + Boost



(d) Despecialization + Boost + Concrete

Figure 2: Plots of immediate reward per episode (left) and cumulative reward per episode (right). These are averages over 30 runs.

	No Respecialization	Respecialization	With Boost	With Boost+Concrete
CATDE	-12.5 | 13.8s	-158.3 | 27.4s	-8.3s | 44.8s	-5.5 | 11.0s
Policy	-20.2 | 18.0s	-45.1 | 1.2s	-30.0s | 96.6s	-11.5 | 11.0s
Value	-5.1 | 19.2s	-7.4 | 1.6s	-5.3s | 48.8s	-5.2 | 14.1s

Figure 3: Final cumulative reward vs average runtimes of our agents.

those choices to result in pathologically poor performance. However, should an agent run out of specializations to attempt, it can cease reevaluating other options, reducing computational overhead.

Our next attempt at a second modification of our agents is to implement a mechanism to boost the likelihood of an agent reselecting specializations that it has selected before. Each time it selects a specialization, it is given a boost that increases quadratically over time: $1 + 2 + \ldots$. This should cause an agent to eventually settle on one of its top choices. Figure 2c depicts the performance of agents that additionally have this *boost* mechanism in place. Given our intuition about why the *blacklist* mechanism was poor, we expected this to have better results, and this is clearly the case. The agents using the CATDE criterion converge better than the agents with no despecialization but at significant computational cost, as highlighted in cyan in figure 3. The *boost* mechanism is generally computationally inefficient since it successfully specializes the value function but never stops reevaluating its decisions even after it has effectively settled on them.

Our final modification is to address the computational overhead of the *boost* mechanism by disabling further attempts to reevaluate whether respecialization should occur at a given level of generality after 300 steps of stability. Figure 2d depicts the performance of agents that additionally have this *concrete* mechanism in place. The performance of these agents more consistently dominates that of the agents with no despecialization. We take this result as evidence supporting our original hypothesis. Additionally, as it reduces computational costs, it's of obvious practical value. These positive results are highlighted in green in figure 3. It's worth noting that while the agents using the value criterion benefit computationally and still do the best of the three criteria, the performance is nonetheless slightly worse than that of the original agents using the value criterion with no respecialization. The robust trend of effective learning using the value criterion would seem to indicate that specialization in order to achieve maximal spread of weights at each level of generality is reliably more effective than either attending to the error experienced by weights or by estimating the number of different actions in which a different policy decision would be made. Our best guess as to why the policy criterion performs poorly is that it is likely effective only when one-step lookahead provides a significant change to the capabilities of the agent.

4 Conclusion

This work implemented much of the future work suggested by Bloch and Laird [2015]. We provide evidence to support Bloch and Laird's [2015] hypothesis that it is more efficient to specialize the value function quickly, making specializations that are potentially suboptimal as a result, and to later modify that value function in the event that the agent gets it "wrong."

The number of distinct variables, however, remains fixed by the features initially provided to the system. Allowing future agents to reason about multiple objects of some type in the environment without needing to enumerate a fixed number of them ahead of time could still be a useful extension for other domains. It would however add another whole dimension to the feature selection problem, effectively resulting in the worst case value function complexity potentially matching environment complexity.

References

[Bloch and Laird, 2013] Mitchell Keith Bloch and John Edwin Laird. Online value function improvement. In *Reinforcement Learning and Decision Making 2013: Extended Abstracts*, pages 111–115, 2013.

[Bloch and Laird, 2015] Mitchell Keith Bloch and John Edwin Laird. The carli architecture–efficient value function specialization for relational reinforcement learning. In *Reinforcement Learning and Decision Making 2015: Extended Abstracts*, pages 44–47, 2015.

[Džeroski *et al.*, 2001] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine learning*, 43(1-2):7–52, 2001.

[Maei and Sutton, 2010] Hamid Reza Maei and Richard S Sutton. Gq (λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the Third Conference on Artificial General Intelligence*, volume 1, pages 91–96, 2010.

[Tadepalli *et al.*, 2004] Prasad Tadepalli, Robert Givan, and Kurt Driessens. Relational reinforcement learning: An overview. In *Proceedings of the ICML-2004 Workshop on Relational Reinforcement Learning*, pages 1–9, 2004.

[Whiteson *et al.*, 2007] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Adaptive tile coding for value function approximation, 2007.

Exploring by Believing

Sara Aronowitz
Department of Philosophy
University of Michigan
Ann Arbor, MI, 48109

Abstract

This paper takes a concept familiar in the decision-making and reinforcement learning literature - the exploration/exploitation trade-off - and extends it to the case of belief. Just as agents should sometimes choose an option with less myopic value, I argue that agents should sometimes believe something which is less likely to be true in order to learn. The question of whether reasons to believe that P can come apart from reasons why P is true has a vexed history in philosophy, but mainly centers on so-called pragmatic reasons to believe, or truth-related reasons based on one-off fantastical scenarios. The contribution of this paper is to use the device of the exploration/exploitation trade-off to posit a systematic epistemic rationale for deviating from a policy of believing all and only what is most likely to be true. Further, features of the trade-off such as a rationale for moving toward exploitation over time, or exploring more during low-reward times, generate interesting and novel results when applied to the belief context. Roughly, the key is that what you believe changes what evidence you acquire, both by changing your behaviors and changing how you imagine and how you search your mental representations.

Keywords: exploration/exploitation trade-off, learning over time, rationality of belief, epistemology

Acknowledgements

Thanks to Reza Hadisi, Sydney Keough, Will Fleischer, Peter Railton, Chandra Sripada, Tim Williamson, and audiences at Athena in Action and the Princeton-Rutgers-Penn Social Epistemology Conference

Introduction

In many decision-making scenarios, we can observe a trade-off between choosing the action that maximizes myopic expected value, or the action most likely to result in learning something new: the **exploration-exploitation trade-off**. For instance, if you were choosing between ordering your favorite ice cream flavor or trying a new one. Exploiting means picking the option most likely, on your estimation, to have the highest value. Exploring, on the other hand, involves choosing something previously untested or about which you're uncertain. There's a trade-off because the best behavior for exploring (say, trying every flavor once, even banana-tomato) is rarely the behavior that is the most likely to maximize reward - and vice versa. The task of this paper is to extend the idea of such a trade-off to the case of belief formation and change: should we ever believe solely in order to explore?

I argue that there is indeed an exploration/exploitation trade-off in belief, because of the connection between our current beliefs and our dispositions to conduct experiments and explore the space of possibilities. While past work has focused on situationally driven trade-offs in epistemic value that are arbitrary and often fantastical[1], this paper posits a systematic and easily implemented rule: deviate occasionally from the recommendations of your best policy in the beginning of inquiry.

This paper proceeds as follows: In § 0.1, I provide background about the multi-armed bandit problem, and describe some features of decisions that are sufficient but not necessary for exploration to be a feature of the optimal strategy. In § 0.2, I note some features of agents who are responsive to the trade-off that will be critical for the case of belief. § 1 provides an informal example of an agent facing a decision about what to believe in which it's rationally permissible for him to explore, or choose an act which is sub-optimal from the point of myopic expected value but provides him with an increased opportunity to learn. The rest of the paper is concerned with connecting features of the original decision problem with features of the belief decision problem.

0.1 The Exploration/Exploitation Trade-Off

In this section, I explain the trade-off through a simplified version of a classic setup in the literature: the multi-armed bandit. I omit this from the abstract.

0.2 Features of the Trade-Off

This section surveys some common features of agents who act via modulating the trade-off.

One feature of the E/E trade-off in action will be crucial for belief: the relationship between E/E and time. As I noted at the end of § 0.1, there's a somewhat generic rationale for preferring to explore more earlier and exploit more later. This reflects a relationship between time and uncertainty - since exploration is more important when uncertainty is high. However, even while holding uncertainty fixed, there's a relationship with time. The information which is reached by exploring has more value when our agent will have a lot more chances to play the slot machines. As she approaches the end of her interaction with the current environment, the diminishing of future opportunities favors exploitation. This is so even if she is still quite uncertain.

Reward changes in the environment also modulate the trade-off. Traca and Rudin [3] show that in environments with periodic reward functions, it's optimal to exploit more during high-reward periods and explore more during low-reward periods. In their case, the varying rewards were due to daily traffic patterns on a website, and at higher traffic times, the recommender algorithm did best by exploiting more, and by exploring more at lower traffic times.

Variations in uncertainty, potential for actions, and total available reward all modulate the exploration-exploitation trade-off in actions. As I now turn to belief, I'll show how each of these factors has a parallel in that case as well.

1 An Example

Consider this description of an incoherent mental state from Meša Selimović's novel *Death and the Dervish.* The context is that the narrator, Sheikh Nurudin, has just passed up an opportunity to mention his incarcerated brother to a powerful woman who might have been willing to help for the right price. The woman has asked for his help in falsely accusing her own brother:

> I had done nothing wrong, but my thoughts were assailed by memories of the dead silence, of the impenetrable darkness and strange, glimmering lights; of the ugly tension and the time I had spent anxiously waiting; of our shameful secrets and thoughts disguised by smiles. I felt as if I had missed something, as if I had made a mistake somewhere, although I did not know where, or how. I did not know. But I was not at peace. I could hardly bear this feeling of uneasiness, this anxiety whose source I could not determine. Maybe it was because I had not mentioned my brother, because I had not insisted that we talk

Figure 1: An epistemological version of the bandit problem, where choosing an arm represents changing your belief state.

> about him. But I had done that on purpose, in order not to spoil my chances. Or was it because I had taken part in an shameful conversation and heard shameful intentions without opposing them, without protecting an innocent man? Only I had had my own reasons, which were more important than all of that, and it would not be right for me to reproach myself too much. For each of my actions I found an excuse, yet my distress remained.

Nurudin is in a conflicted state, he thinks he's done nothing wrong... but in another sense, knows that he has (“I had done nothing wrong" but “I felt as if I had missed something”[1]). Furthermore, while he is in a situation of moral urgency (figuring out what to do about his brother, as well as figuring out if he is being a good person), it's not a conflict between truth and morality. Rather, it's morally significant to figure out the truth. In response to this conflict, he adopts a divided mental state, where the two conflicting attitudes play somewhat different functional roles. That is, his thought that he has made a mistake is evidenced by his fervid rumination on the past events, searching for where he went wrong. On the other hand, his thought that he did nothing wrong is evidenced by his conscious belief and more in line with his previous beliefs and convictions.

> **Conflicted Nurudin**: Nurudin is receiving conflicting evidence about how to live that may contradict his deeply held convictions about a life of disconnection with material things. He responds by taking an incoherent stance: he still believes in his former convictions and uses them to make inferences about morality, to give advice, and partially to plan for the future. At the same time, he starts to ruminate and even act in accordance with a different set of beliefs, on which his way of life is foolish and it is morally unacceptable to attempt disconnection with the world.

As it turns out in the book, Nurudin is at the beginning of a slow shift from being withdrawn from the world, comfortable in a rigid belief system to getting involved in an insurrection and filled with doubt about everything. But this drift from one state to the other takes up the whole book (which occurs over something like a year in real time) and in this period, he is deeply incoherent.

We could imagine someone like Nurudin who flips his belief switch between one context and the next - maybe he's a true believer when in the tekija (a monastery-like place where the dervishes live), and full of doubt when moving about the city. But this is clearly not the state being described above[2].

Now, since Nurudin is so divided, we might have trouble even generating a measure for his predicted accuracy. Formally, we know that no proper scoring rule will assign the highest possible accuracy to a set of beliefs containing a contradiction. Informally, he just can't be entirely right about his own misdeed if he believes two opposing things, so I think it's reasonable to suppose that he's in a suboptimal epistemic state, and that this is accessible to him. This means that by adopting the strategy I've described, Nurudin is by definition taking a hit in myopic epistemic value.

Further, Nurudin presumably does not plan on being in this divided state forever; instead, he's developing two incoherent projects in parallel in order to eventually be able to figure out which is better. Since the belief in the Sufi system is one that comes with an entire moral and descriptive framework, it's reasonable to think that either future coherent state will have very different standards of evaluation, and recommend distinct experiments. So Nurudin is also in a state of meta-uncertainty, which he responds to by moving to a less accurate state (in this case by *all* standards) that might improve his prospect of learning.

2 Argument for a belief/action symmetry

In this section, I argue that there is no relevant difference between the standard bandit problem and the case of Nurudin that would suggest the exploration/exploitation trade-off is not part of the correct description of his situation. More generally, I present a view on which the trade-off should be a normal part of our reasoning about what to believe.

To make this argument, I'll sketch the features of the original case, and then extend them to Nurudin's. It's important to note here that parity between the bandit and Nurudin is a stronger criterion than necessary to establish the existence

---

[1] in fact, “I felt" is a translation of “činilo mi se" in the original, which does not connote anything particularly emotional

[2] In fact, a different version of this paper would use the strategy of oscillating back and forth between beliefs as an example of an exploration behavior. The key here is that regular oscillations of this kind are not triggered by evidence, but are features of a broader strategy. Plausibly, the justification for such a strategy mirrors the one I'll give here - it represents a hedge between the two other options intended to optimize evidence-gathering.

of an exploration/exploitation trade-off in belief; multi-armed bandits are one of many problems whose solutions exhibit this trade-off ranging from tree search to applied problems in robotics.

Here's an overview of those features:

	action (standard) bandit	*belief bandit*
1) act generates evidence:	usually	usually
2) act results in reward:	sometimes	sometimes
3) evidence and reward come apart:	usually	sometimes
4) procedure is iterated:	approximately	approximately

The original case exhibited a trade-off because we expect pulling the lever to give us evidence about the underlying function, and also a reward. These features come apart, as in the case of a high-reward lever that has been pulled many many times. And finally the process needs to be iterated - otherwise exploitation would always trump exploration. I aim to show that all of these features are shared with belief: belief changes what kinds of evidence we can expect to receive based on our dispositions to imagine and conduct experiments, it gives us a 'reward' in the form of accurate beliefs, and these two can come apart in cases like Nurudin's. Finally, each act of forming a belief, like each action, is in some sense perfectly unique - but in both cases, we're engaged in a complex process that can be approximated for some purposes by treating it as an iteration of the same moves in different contexts and orders.

3 Conditions 1,2 and 4

Conditions 1 and 2 are easily satisfied by belief. For 1, beliefs lead to the acquisition of evidence through experimentation and imagination. For 2, f we treat accuracy as a kind of epistemic reward, as is commonly done in formal epistemology, beliefs are predictably evaluable for reward. Condition 4 is somewhat more complicated, but I argue that it holds because the simplifications necessary to treat belief as an iterated problem are of a kind with the simplifications necessary in the case of action - in neither case is there literal repitition, but it's close enough for the idealization to be useful in action, so it should also be for belief.

4 Condition 3

This condition is doing most of the work, so I treat it in more depth. I present two connections between belief and other acts that result in a gap between the expected reward (i.e. accuracy or truth) of a package of beliefs, and the expected evidential value (i.e. likelihood to lead to new and/or significant information being gleaned). The first connection is a familiar one between belief and experimentation, and the second is the more rarely discussed connection between belief and imaginative search. In addition to demonstrating that condition 3 obtains, I aim to uncover what features and limitations an agent and her environment must have in order for this condition to obtain.

4.1 Belief and Evidence-Gathering

Evidence-gathering by definition opens up the prospect of changing beliefs and evidential standards; when we plan on getting evidence, we plan on changing our beliefs. This feature seems like enough to allow a rational agent to prefer exploration in the practice of getting evidence at least - but can you vary evidence-gathering practices without changing current beliefs, and in particular, without giving up some current predicted accuracy?

Maybe all we need is to accept some helpful propositions rather than change our beliefs. An intuitive response to Nurudin's case is that the best thing for him to do would be to accept the two incoherent propositions about moral living, rather than believe them. This is presumably what would be recommended by Van Fraassen [4] among others.

What is meant by acceptance? On one end of the spectrum is the view that acceptance just involves acting as if something were the case without changing any beliefs or making any new inferences. On the other end is the idea that acceptance is descriptively identical to belief, it just falls under different normative standards. The latter in this context is question-begging, since I'm arguing that the normative standards for belief should cover exploration as well. Likewise for the suggestion that acceptance and belief differ only insofar as beliefs are only sensitive to direct, truth-seeking considerations (see Shah and Velleman [2]). So I'll consider two cases, one where acceptance is just acting-as-if, and the other where it involves some doxastic changes but not as far as in belief (perhaps inferences drawn from the accepted proposition are quarantined away from the agents other beliefs, for instance).

In the first case, if Nurudin accepts and does not accept the Sufi moral system, but keeps his .5 credence, there are two issues. First, it's not clear that it's even possible to act as if Sufism is correct and incorrect- acting does not allow for synchronic partitions the way belief does. Second, given some plausible psychological postulates, the advantages that Nurudin gets from believing will not transfer to acting-as-if. For instance, it was his wholehearted investment in Sufism

points of contact between belief and imaginative search

1. Starting points: Imagination often involves starting with our current beliefs about the world, and departing from them incrementally. So in general, we explore neighboring theories first.
2. Side-constraints: Imaginative exercises often involve coming up against previously implicit limits - for instance, in thought experiments when we observe an unanticipated reaction in ourselves that shapes the way we fill in a case.
3. Goals: Imaginative search is a combination of pure random wandering and goal-oriented activity, and the goal-directed end of the continuum depends on beliefs about what's valuable and how to achieve those ends.
4. Stopping rules: For a computationally limited group, or one interested in computational efficiency, it will be necessary to regulate the amount of search, for instance allocating more time to search when the group seems to have reached a plateau or less when current possibilities overwhelm resources to pursue confirmation of those possibilities.
5. Costs and cost analysis: Some theories cost more resources to build than others - another consideration that favors nearby possibilities.

Figure 2: Sketch of possible relations between beliefs and imaginative search

that allowed him to stay up late studying; if he was motivated by his belief, and acceptance is just acting-as-if, then we have nothing to replace that motivation. Treating acceptance as a mere pattern of action by definition does not give us any resources to explain motivation.

On the other hand, let's say acceptance involves some but all of the internal properties of belief. Which features might then separate the two? One idea is that acceptance might be conditional, ready to be taken back when necessary; we might want to make sure to keep track of all the inferential consequences of an accepted proposition. I'll come back to this in the next section.

In general, the connection between belief and evidence-gathering gives us a reason to be willing to take an accuracy hit in order to explore. Agents like us are motivated to gather evidence that we judge to be promising, interesting or fruitful based on our other beliefs. These experiments form epistemic projects, and can be said to rely on a shared basis of belief in which we need to be invested in order to be motivated to gather the relevant evidence. However, even if this motivational structure were not in place, we might expect that rational agents would be constrained in their experimentation by their beliefs. Even if we place acceptance between belief and evidence-gathering in some cases, this relationship will still persist, albeit indirectly, since beliefs will affect acceptance which will affect evidence-gathering. Thus some of our beliefs can still be evaluated based on their consequences for acquiring evidence, though these consequences will be somewhat indirect.

To give up on a *normative* connection between what we believe and what evidence we collect is to invest in an epistemology that starkly separates evidential response from inquiry. This move should be a kind of last resort, since it involves relinquishing a plausible route to explaining why experiments are justified. In the next section, I'll argue that the creation of a hypothesis is a kind of internal inquiry that depends on beliefs but carries consequences for future learning.

4.2 Consideration and New Hypotheses

Imagine that Nurudin has an unlimited capacity to control and shape his own motivational faculty, and let's add that Van Fraassen was right, and he should accept, rather than believe, the existence and non-existence of God. He'll go ahead and behave as if he doesn't believe in God during the day, and does at night, but he will be in a state of suspension of belief.

This Nurudin, I'll argue, will face a dilemma when it come to constructing new hypotheses. Normally, for an agent who isn't logically omniscient, imagination is one route to forming new theories. In particular, we don't imagine totally random possibilities out of the blue, but we use our current understanding of how things actually are, together with beliefs about what is and isn't possible, to come up with new options. For instance, he might start by visualizing an atom according to his current theory, and sort of play around with the possible shapes (according to his ideas about what shapes can do (innate or otherwise)). By this process, he comes up with a few alternative ideas.

In fig. 2, I describe some of the possible constraints imposed by belief.

These features are related to the truth of a belief, but only indirectly. That is, one set of beliefs S could confer more predictable advantage according to the values in fig. 2 than another set of beliefs T without S having a higher estimated accuracy than T. So Condition 3 holds for belief - truth value and evidential value can come apart (i.e. they need not coincide to the same degree in all situations).

5 Objections

One significant challenge is that if we accept the rationality of the exploration-exploitation trade-off in action, positing an additional trade-off for belief amounts to two solutions to one problem, where each solution is on its own sufficient. That is, isn't introducing exploration twice overkill?

There's something undoubtibly correct in this suggestion - agents who introduce randomness or other exploration behaviors at multiple points face a difficulty in making sure these interventions are consistent. In some situations, introducing exploration at the level of action will be enough to reduce the agent's chance of getting stuck in a local maximum. And likewise for imaginative search; if we introduce randomness into the search process itself, that will solve some of the problems of a purely exploitative approach.

However, this will not always be the case, and there are benefits to belief exploration which do not carry over to imaginative exploration. Consider how it is that Nurudin's beliefs allow him to explore in figure **??**. It's not just that he happens to explore theories that are adjacent to his beliefs; these theories are *made more accessible* to him by his beliefs. Because he believes in Sufism, through coordination of actions, imagination and other modes of thought, he's amassed resources to understand that theory and how it might be altered to create new versions. For one not familiar with Sufism in that intimate, thorough-going way, it wouldn't be clear, for instance, that there are two versions of the view, one which takes the mystical state of oneness with God to have content, and one which doesn't. Given this, in order to gain the advantage of the incoherent package by only changing actions, there would need to be a coordinated exploratory change to both external actions and imaginative ones. Changing the underlying beliefs is a natural and effective way of achieving this coordination. Further, even changing external actions and imagination in a coordinated way would likely be insufficient; part of how belief makes regions of possible space accesible is intrinsic, coming from the fact that believing in something involves entertaining that proposition fully, in a way that seems deeper than other forms of non-doxastic consideration.

6 Conclusion

A country song asks:"how am I ever gonna get to be old and wise, if I ain't ever young and crazy?". In this paper, I argue that this same line of thought applies to belief - in the beginning of inquiry, we should believe in order to explore rather than to exploit, but as inquiry progresses, we should drift towards maximizing accuracy value. This is a feature shared between action and belief, and exploits the rational connection between belief and imagination.

This paper aims to answer a philosophical question - what kinds of things can be reasons to believe - but in a way that utilizes formal devices from other fields. But I think it should be of interest to researchers in learning and decision theory as well, since these fields also use formal machinery for belief as well as for action. Using this framework, there are potentially interesting interactions between modulating E/E in belief and in action in an agent who both wants to gain external reward and also wants to be right.

References

[1] Hilary Greaves. "Epistemic decision theory". In: *Mind* 122.488 (2013), pp. 915–952.
[2] Nishi Shah and J David Velleman. "Doxastic deliberation". In: *The Philosophical Review* (2005), pp. 497–534.
[3] Stefano Tracà and Cynthia Rudin. "Regulating Greed Over Time". In: *arXiv preprint arXiv:1505.05629* (2015).
[4] Bas C Van Fraassen. *The scientific image*. Oxford University Press, 1980.

Discovering Symmetries for Sample Efficient Reinforcement Learning

Anuj Mahajan
Conduent Labs India
Bangalore, India
anujmahajan.iitd@gmail.com

Theja Tulabandhula
University of Illinois Chicago
Chicago, USA
tt@theja.org

Abstract

With recent advances in the use of deep networks for complex reinforcement learning (RL) tasks which require large amounts of training data, ensuring sample efficiency has become an important problem. In this work we introduce a novel method to detect environment symmetries using reward trails observed during episodic experience. Next we provide a framework to incorporate the discovered symmetries for functional approximation to improve sample efficiency. Finally, we show that the use of potential based reward shaping is especially effective for our symmetry exploitation mechanism. Experiments on classical problems show that our method improves the learning performance significantly by utilizing symmetry information.

Keywords: Deep Reinforcement Learning; Functional Approximation; Symmetry.

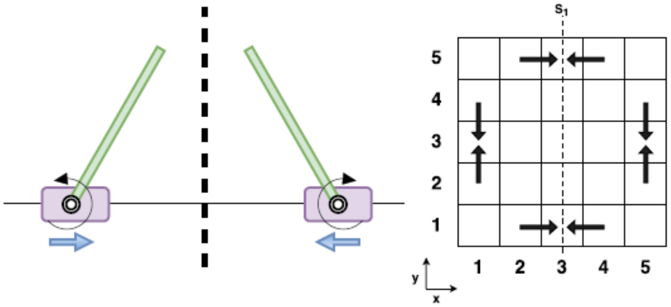


Figure 1: *Left*: Symmetry in Cart-Pole environment. *Right*: Symmetry in grid world. $f((x,y)) = (6-x, y)$, $g_s : \{N \to N, S \to S, E \to W, W \to E\} \forall s$.

1 Introduction

In many RL scenarios, like training a rover to move on Martian surface, the cost of obtaining samples for learning can be high (in terms of robot's energy expenditure), and so sample efficiency is an important subproblem which deserves special attention. Very often the environment has intrinsic symmetries which can be leveraged by the agent to improve performance and learn more efficiently. Moreover, in many environments, the number of symmetry relations tend to increase with the dimensionality of the state-action space, e.g. for the simple case of grid world of dimension d there exist $O(d!2^d)$ fold symmetries. This can provide substantial gains in sample efficiency while learning as we would ideally need to consider only the equivalence classes formed under the induced symmetry relations. However, discovering these symmetries can be a challenging problem owing to noise in observations and complicated dynamics of the environment. With recent advances in deep reinforcement learning [5, 4], it has been demonstrated that a lot of seemingly complex tasks which pose challenges in the form of large state action spaces and difficulty of learning good representations [1, 8]; can be handled very well with the use of deep networks as functional approximators. Training these networks, however requires large amounts of data and learning their parameters necessitates coming up with careful update procedures and defining the right objectives(see Glorot et al [3]) and is a topic of study in its own right. Thus established methods for MDP abstraction and minimization can't be practically extended for use in function approximation for RL using deep networks which builds up the premise for this work. To the best of our knowledge, we are the first to motivate the use of symmetry in the context of deep reinforcement learning. In this paper we investigate methods for discovering state space symmetries and their inclusion as prior information via a suitable cost objective.

2 Preliminaries

Symmetries in MDP: We represent MDP as a tuple $M := \langle S, A, \Psi, T, R \rangle$, here $\Psi \subset S \times A$ is the set of admissible state-action pairs, $R : \Psi \times S \to \mathbb{R}$, $T : \Psi \times S \to [0,1]$ are reward and transition functions respectively. The notion of symmetries in MDP can be rigorously treated using the concept of MDP *homomorphisms*[7]. MDP homomorphism h from $M = \langle S, A, \Psi, T, R \rangle$ to $M' = \langle S', A', \Psi', T', R' \rangle$ is defined as a surjection $h : \Psi \to \Psi'$, which is itself defined by a tuple of surjections $\langle f, \{g_s, s \in S\} \rangle$. In particular, $h((s,a)) := (f(s), g_s(a))$, with $f : S \to S'$ and $g_s : A_s \to A'_{f(s)}$, & satisfies following: Firstly it preserves the rewards (i.e., $R'(f(s), g_s(a), f(s')) = R(s, a, s')$) & secondly it commutes with dynamics of M (i.e., $T'(f(s), g_s(a), f(s')) = T(s, a, [s']_{B_{h|S}})$). Here we use the notation $[\cdot]_{B_{h|S}}$ to denote the *projection* of equivalence classes B that partition Ψ under the relation $h((s,a)) = (s', a')$ on to S. Symmetries $\chi : \Psi \to \Psi$ can then be formally defined as *automorphisms* on M with the underlying functions f, g_s being bijective. The homomorphism requirements for a symmetry reduce to:

$$T(f(s), g_s(a), f(s')) = T(s, a, s') \tag{1}$$

$$R(f(s), g_s(a), f(s')) = R(s, a, s') \tag{2}$$

The set of *equivalence classes* $C[(s,a)]$ of state action pairs formed under the relation $\chi((s,a)) = (s', a')$ partition Ψ and can thus be used to form a quotient MDP, represented as $M_Q = M/C$, which is smaller and can be efficiently solved. However in RL setting, we do not know the underlying system dynamics, hence we estimate $C[(s,a)]$ on the go using the agent's experience and use estimated equivalent classes to drive identical updates for equivalent state-action pairs during the process of learning the optimal policy.

Reward Shaping: Reward shaping is a technique that augments the reward function R of a MDP $M = \langle S, A, \Psi, T, R \rangle$ by a shaping function $F : \Psi \times S \to \mathbb{R}$. Thus, the agent sees a modified reward $R'(s, a, s') = R(s, a, s') + F(s, a, s')$. Ng et al. [6] have shown that if the reward shaping function is potential based i.e., is of the form: $F(s, a, s') = \gamma\Theta(s') - \Theta(s) \ \ \forall s, a, s'$

Table 1: Episodes to convergence(rounded) : Grid-World

Setup	Episode length	Naive	Girgin	Sym
9x9 Θ_1	480	136 ± 7	113 ± 5	$\mathbf{37 \pm 6}$
13x13 Θ_1	800	237 ± 6	166 ± 7	$\mathbf{46 \pm 6}$
9x9 Θ_2	480	174 ± 7	116 ± 6	$\mathbf{31 \pm 5}$
13x13 Θ_2	800	241 ± 5	187 ± 6	$\mathbf{38 \pm 6}$
5x5x5 Θ_2	1000	253 ± 8	177 ± 6	$\mathbf{41 \pm 7}$
7x7x7 Θ_2	1500	275 ± 9	229 ± 10	$\mathbf{54 \pm 8}$

for some $\Theta : S \rightarrow \mathbb{R}$, then the policy invariance property is guaranteed. Shaping helps distinguish these pairs by making the rewards sufficiently distinct and consequently preventing spurious similarity estimates.

3 Methodology

Symmetry Detection: Given an MDP $M = \langle S, A, \Psi, T, R \rangle$, we define set $\Pi_{sa,j} = \{(\sigma, N_\sigma)\}$ where σ is a sequence of rewards of length j and N_σ is the number of times it is seen starting with state s taking action a during the execution of policy π. We use the notation $|\Pi_{sa,j}| = \sum_{|\sigma|=j} N_\sigma$ and $\Pi_{sa,j} \cap \Pi_{s'a',j} = \{(\sigma, \min(N_\sigma, N'_\sigma))\}$. We define the notion of similarity between two state action pairs $\langle s, a \rangle$ and $\langle s'a' \rangle$ as follows:

$$\chi_{i,l_0}(\langle s,a \rangle, \langle s'a' \rangle) = \frac{\sum_{j=l_0}^{i} |\Pi_{sa,j} \cap \Pi_{s'a',j}|}{(\sum_{j=l_0}^{i} |\Pi_{sa,j}| * \sum_{j=l_0}^{i} |\Pi_{s'a',j}|)^{1/2}} \tag{3}$$

To efficiently compute the similarities between all the state action pairs, we use an auxiliary structure called the reward history tree P, which stores the prefixes of reward sequences of length up to i for the state action pairs observed during policy execution and also maintains a list of state, action, occurrence frequency tuples $[\langle s, a, o \rangle]$. The similarities estimates can be computed by doing a breadth first traversal on P. We consider state pairs

$$\chi_{sym} := \{\langle s,a \rangle, \langle s'a' \rangle | \chi_{i,l_0}(\langle s,a \rangle, \langle s'a' \rangle) \geq \Delta\}$$

as similar for the given length (i and l_0) and threshold parameters (Δ). Note that the definition of χ_{i,l_0} enables finding state action symmetries even when the actions are not invariant under the symmetry transform ie. $g_s(a) \neq a \forall s, a$ (indeed, this is the case with the Cart-Pole problem where $\forall s \in S$, $g_s(Left) = Right, g_s(Right) = Left$). Previous work in [2] is unable to do this. It can be shown that the similarity measure $\chi_{l_0,i}$ is complete ie. any state action pair which is equivalent under the given symmetry should be identifiable using 3.

Symmetry Inclusion Priors & Learning: Let $Q(s, a; \theta)$ be the function approximator network. Having found some symmetric state action pairs χ_{sym}, we next use this information while training the network.

$$L_{i,total}(\theta_i) = L_{i,TD}(\theta_i) + \lambda L_{i,Sym}(\theta_i), \tag{4}$$

Eq. 4 gives training loss, where λ is weighing parameter and the loss components are:

$$L_{i,TD}(\theta_i) = \mathbb{E}_{\mathbb{B}}[((r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})) - Q(s, a; \theta_i))^2] \tag{5}$$

$$L_{i,Sym}(\theta_i) = \mathbb{E}_{\chi_{sym}}[(Q(s', a'; \theta_i) - Q(s, a; \theta_i)^2] \tag{6}$$

Here $\mathbb{B}$ is the set of observed (s, a, r, s') tuples following a ϵ-greedy behavioral policy. Differentiating the total loss (4) with respect to the weights, we arrive at a combination of gradients coming from the two loss objectives:

$$\nabla_{\theta_i} L_{i,TD}(\theta_i) = \mathbb{E}_{\pi,s}[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i)] \tag{7}$$

$$\nabla_{\theta_i} L_{i,Sym}(\theta_i) = \mathbb{E}_{\pi,\chi_{sym}}[(Q(s', a'; \theta_i) - Q(s, a; \theta_{i-1}))\nabla_{\theta_i} Q(s', a'; \theta_i)] \tag{8}$$

Eq. 7 represents the familiar Q-learning gradient. Eq. 8 is defined so to prevent the network from destroying the knowledge gained from current episode. A symmetric version of the DQN algorithm is given below.

4 Experiments

Grid World: In the grid world domain we consider the problem of finding the shortest path to a goal in a $[1..n]^d$ sized grid world. The discount factor is set to be $\gamma = 0.9$ and exploration $\epsilon = 0.1$. The goal state(x_G, y_G) is chosen randomly at the start of each iteration. We test two kinds of reward shaping settings $\Theta_1(x, y) = (|x - x_G| + |y - y_G|)$, $\Theta_2(x, y) = (|x - x_G| + |y - y_G|)\gamma^{|x - x_G| + |y - y_G|}$. Table 1 gives the number of episodes required for convergence to optimal policy

Algorithm 1 Sym DQN

```
Initialize: Memory $\mathbb{D} \leftarrow \{\}, P \leftarrow \{\{root\}, \{\}\}$
Initialize action-value function Q with random weights($\theta$)
for $episode \leq M$ do
    Initialize start state
    for $t = 1$ to $T$ do
        With probability $\epsilon$ select action $a_t$
        Otherwise select $a_t = argmax_a Q(s_t, a, \theta)$
        Execute action $a_t$ and observe reward $r_t$ state $s_{t+1}$
        Store transition $(s_t; s_t; r_t; s_{t+1})$ in $\mathbb{D}$
        Sample random minibatch $\mathbb{B}$ from $\mathbb{D}$
        Find $\mathbb{B}_s$ the batch of symmetric pairs of $\mathbb{B}$ from $P$
        Set targets for $\mathbb{B}$ & $\mathbb{B}_s$
        Perform gradient descent step as in eq 7,8
    Update $P$ with the episode
```

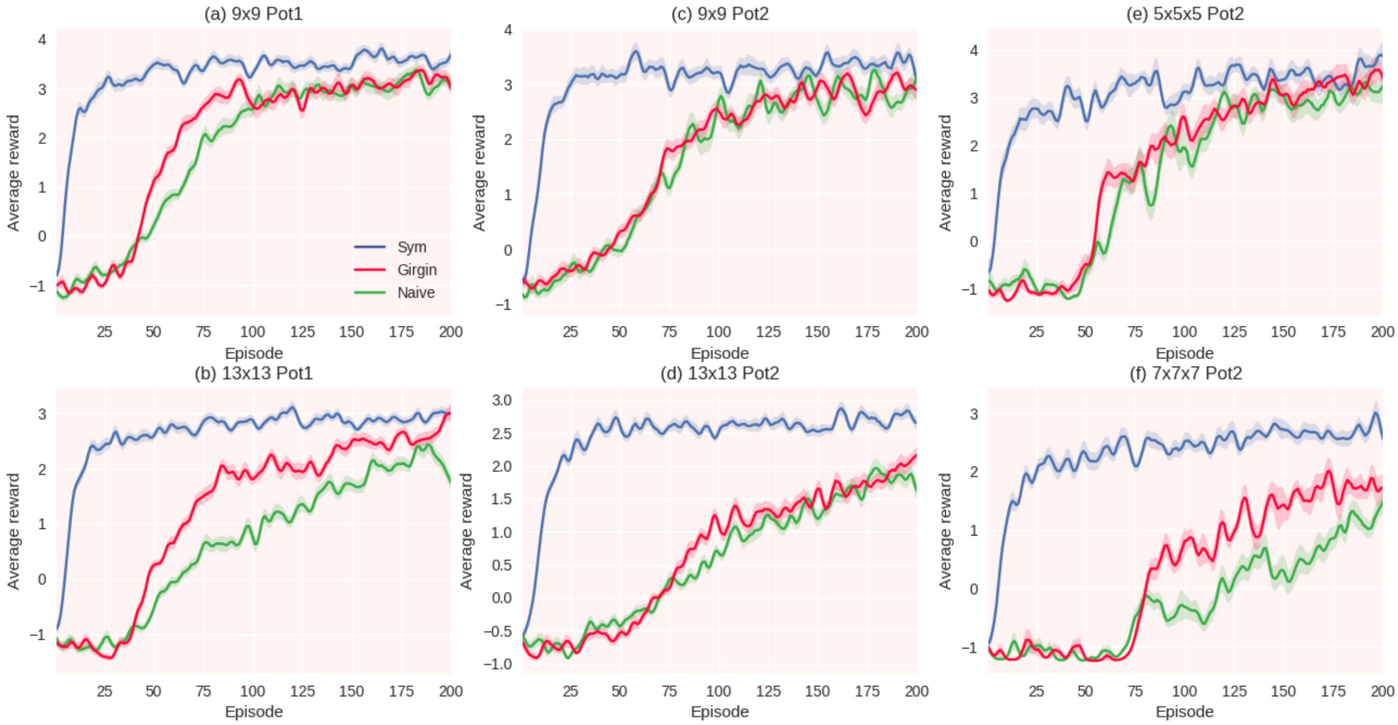


Figure 2: Grid-World plots with one standard deviation errors. $\Delta = 0.8, l_0 = 1, i = 5, \lambda = 0.4$. No of iterations: 50.

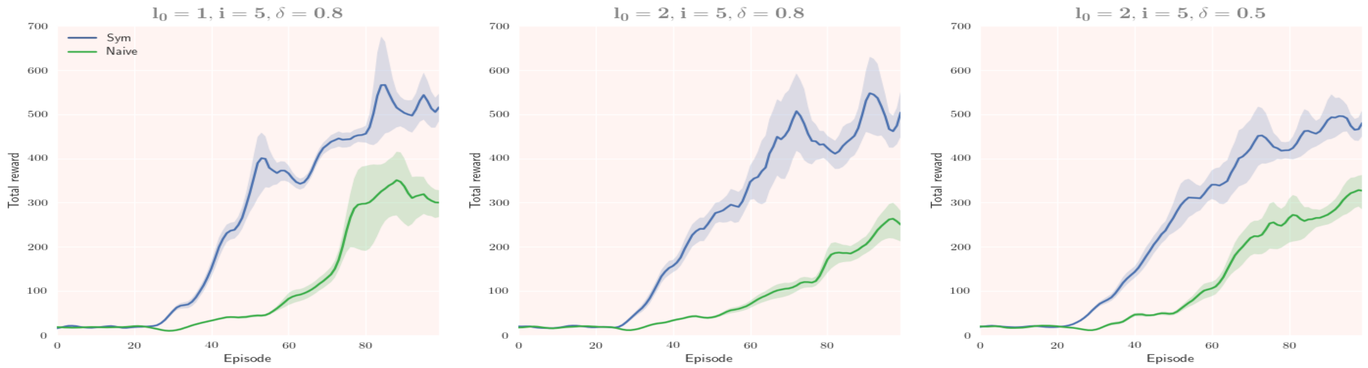


Figure 3: Total reward with episodes. All experiments used $\lambda = 1, \gamma = 0.99, \epsilon$ was decayed from 1 to 0.1 rate 0.98.

Table 2: Performance and statistical significance : Cart-Pole Domain

			mean			**max**		
l_0	**i**	**Δ**	**DQN**	**SymDQN**	**P-value**	**DQN**	**SymDQN**	**P-value**
1	5	0.8	102.83 ± 69.74	$\mathbf{262.74 \pm 64.59}$	$< 10^{-5}$	537.38 ± 155.89	$\mathbf{796.23 \pm 173.65}$	2×10^{-4}
1	5	0.5	93.47 ± 62.16	$\mathbf{233.92 \pm 69.48}$	$< 10^{-4}$	481.60 ± 163.25	$\mathbf{812.07 \pm 181.91}$	$< 10^{-4}$
1	4	0.8	107.96 ± 67.17	$\mathbf{236.14 \pm 76.12}$	$< 10^{-4}$	561.27 ± 156.97	$\mathbf{759.36 \pm 168.32}$	2.4×10^{-3}
2	5	0.8	81.57 ± 53.92	$\mathbf{232.65 \pm 89.73}$	$< 10^{-4}$	514.53 ± 161.58	$\mathbf{860.17 \pm 192.71}$	$< 10^{-4}$
2	5	0.5	115.24 ± 57.35	$\mathbf{257.02 \pm 58.49}$	$< 10^{-4}$	467.92 ± 183.36	$\mathbf{671.87 \pm 169.37}$	3.7×10^{-3}
2	4	0.8	91.33 ± 71.39	$\mathbf{229.48 \pm 80.56}$	$< 10^{-4}$	503.76 ± 162.13	$\mathbf{807.49 \pm 177.18}$	$< 10^{-4}$

for each setup in Figure 2. The agent running our algorithm (labeled Sym) learns the optimal policy much faster than baseline (labeled $Naive$) and previous work. On performing the two sided Welsh's t-test on the null hypothesis the p-values are found to be $< 10^{-4}$ for all the setups.

Cart-Pole: The agent's state is given by (θ, x, ω, v) denoting the angular position of pole, position of cart and their time derivatives. The action space is $\{Left, Right\}$ for the cart. We define a reward shaping function as follows: the state space is uniforly discretized into $L = 9$ levels along each dimension. For bounding box $[-X_b, X_b]$, the interval width is $w = \frac{2X_b}{L}$ and the discrete dimension is $x_d = k$ if $\frac{(2k+1)w}{2} \geq x \geq \frac{(2k-1)w}{2}$. The shaping function is defined as : $F(\theta, x, \omega, v) = 1 - \frac{(\theta_d^2 + x_d^2 + \omega_d^2 + v_d^2)}{(L-1)^2}$. The algorithms we experimented with are the DQN [5] agent and its proposed symmetric variant Symmetric DQN (Algorithm 1) agent. The maximum episode length is set to 1500 the replay memory size is set 100000 and the mini batch size of 128 is used. Agents are run under a completely random policy for the first 25 episodes to ensure proper initialization of the replay memory. Figure 3 shows the variation of total reward obtained with the number of episodes averaged over 15 iterations for three different parameter settings. Table 2 gives the mean and the maximum values of the total rewards obtained in an episode for the two algorithms, SymDQN clearly performs much better in both the metrics.

5 Conclusions

We have proposed a novel framework for discovering environment symmetries and exploiting them for the paradigm of function approximation. The experiments verify the effectiveness of our approach. For future work we would like to explore methods for learning the symmetries using deep networks.

References

[1] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[2] S. Girgin, F. Polat, and R. Alhajj. State similarity based approach for improving performance in RL. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 817–822, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[3] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[6] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: theory and application to reward shaping. In *International Conference on Machine Learning*, volume 99, pages 278–287, 1999.

[7] B. Ravindran and A. G. Barto. Symmetries and model minimization in markov decision processes. Technical report, Amherst, MA, USA, 2001.

[8] D. Williams and G. Hinton. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Learning against sequential opponents in repeated stochastic games

Pablo Hernandez-Leal
Centrum Wiskunde & Informatica
Science Park 123, Amsterdam, The Netherlands
`Pablo.Hernandez@cwi.nl`

Michael Kaisers
Centrum Wiskunde & Informatica
Science Park 123, Amsterdam, The Netherlands
`Michael.Kaisers@cwi.nl`

Abstract

This article considers multiagent algorithms that aim to find the best response in strategic interactions by learning about the game and their opponents from observations. In contrast to many state-of-the-art algorithms that assume repeated interaction with a fixed set of opponents (or even *self-play*), a learner in the real world is more likely to encounter the same strategic situation with changing counter-parties. First, we present a formal model of such *sequential interactions*, in which subsets from the player population are drawn sequentially to play a repeated stochastic game with an unknown (small) number of repetitions. In this setting the agents observe their joint actions but not the opponent identity. Second, we propose a learning algorithm to act in these sequential interactions. Our algorithm explicitly models the different opponents and their switching frequency to obtain an acting policy. It combines the multiagent algorithm PEPPER for repeated stochastic games with Bayesian inference to compute a belief over the hypothesized opponent behaviors, which is updated during interaction. This enables the agent to select the appropriate opponent model and to compute an adequate response. Our results show an efficient detection of the opponent based on its behavior, obtaining higher average rewards than a baseline (not modelling the opponents) in repeated stochastic games.

Keywords: Multiagent learning;
reinforcement learning;
repeated stochastic games

Acknowledgements

This research has received funding through the ERA-Net Smart Grids Plus project Grid-Friends, with support from the European Union's Horizon 2020 research and innovation programme.

1 Introduction

Learning to act in multiagent systems has received attention mainly from game theory, focused on algorithms that converge to the Nash equilibrium [2, 6, 7]; and from reinforcement learning, focused on acting optimally in stochastic scenarios with limited a priori information but online observations [5, 11]. However, results are typically based on the assumption of self-play [3, 7], i.e., all participants use the same algorithm, and a long period of repeated interactions.

In contrast, we focus on *sequential interactions* (as a variation of repeated stochastic games), i.e., the agent is paired with stochastically drawn opponents, with whom the agent interacts in short periods while observing joint actions, but without observing the opponent's identity. This article tackles the problem of finding the best response in this setting, which requires to hypothesize possible opponent types, and to identify opponents from their behavior. We propose an algorithm that supposes and identifies several common reasoning strategies, and which best-responds in face of residual uncertainty.

Related work in repeated stochastic games has proposed different learning algorithms [8, 9]. However, it is an open problem how to act quickly and optimally when facing different opponents, especially when uninformed about such changes. Bayesian and type based approaches are a natural fit for this kind of setting. A general and highly complex model are I-POMDPs [12] which are a framework for decision making in uncertain multiagent settings. Bayesian Policy Reuse (BPR) [15] is a framework for acting quickly in single-agent environments, assuming the availability of a policy library and prior knowledge of the performance of the library over different tasks. BPR computes a belief over *tasks* that is updated at every interaction and which is used to select the policy that maximises the expected reward given the current belief. Similarly, in this article the agent maintains believes over *opponent models*. Recently, work on Stochastic Bayesian Games [1] has compared several ways to incorporate observations into beliefs over opponent types when those types are re-drawn after every state transition. Our approach is similar in believe structure, but in contrast assumes that opponents are only redrawn after several repeated stochastic games.

Changes in the environment are also modeled explicitly by learning algorithms for non-stationary environments. Their goal is to learn an optimal policy and at the same time detect when the environment has changed, updating the acting policy accordingly. Reinforcement Learning with Context detection (RL-CD) [10] works in single agent tasks with a changing environment, i.e., different *contexts*. RL-CD learns a model of the specific task and assumes an environment that changes infrequently. To detect a new context RL-CD computes a quality measure of the learned models. Similar problems have been tackled in two-player repeated normal-form games, assuming an opponent with different stationary strategies to select from during a repeated interaction [13, 14]. The learning agent needs to learn online how to act optimally against each strategy while detecting when the opponent changes to a different one. While these works might be the closest state of the art, these approaches do not consider repeated stochastic games, which are central to our discussion.

We contribute to the state of the art with a framework that we name Opponent Learning in Sequential Interactions (OLSI), for best responding in repeated stochastic games by maintaining a hypothesized opponent model set. We compute a belief over this set, updating the belief and the set based on observations. The belief provides a way to compute future expected rewards more accurately and improves empirical performance, in our evaluation approaching optimality.

2 Problem setting

In contrast to classical RL, which considers one single agent in a stationary environment, Game theory studies rational decision making when several agents interact in a strategic conflict of interest, formalized as a *Game*. Note that different areas provide different terminology. Therefore, we will use the terms player and agent interchangeably and we will refer to other agents as opponents irrespective of the domain's or agent's cooperative or adversarial nature.

Our exposition of the approach is build on games with two players i and $-i$. A stochastic game comprises a set of stage games S (also known as states). In each stage s all players choose their part of the joint action $\mathbf{a} \in A(s)$. A game begins in a state $s_b \in S$. A joint action $\mathbf{a} = (a_i, a_{-i})$ is played at stage s and player i receives an immediate reward $r_i(s, \mathbf{a})$, the world transitions into a new stage s' according to the transition model $T(s, s', \mathbf{a})$. Reaching any goal state $s_g \in S$ terminates the game. The accumulated reward of a game is called an *episodic* reward.

We formalize *Sequential Interactions* (SI) as a specific variation of repeated stochastic games, where at each episode $k \in \{1, 2, \ldots, K\}$ a process draws a set of players $P_k \subset I$ from the population of individuals I to play a finite stochastic game that yields a reward (accumulated over the game) to each player. After the stochastic game terminates, the subsequent interaction commences. We specifically discuss the setting where the selection process is *stochastic* (as opposed to being a *strategic choice* by the agents), and the population comprises an unknown distribution over types of strategies. We consider P_k and opponent rewards within the stochastic game to be unobservable, while the joint actions are observable. In addition, neither $r_i(s, \mathbf{a})$ nor $T(s, s', \mathbf{a})$ are known. In our approach, the agent learns explicit models of $r_i(s, \mathbf{a}), T(s, s', \mathbf{a})$ and P_k from observations, and may update its hypothesized model of I.

3 Opponent learning in sequential interactions

Our approach is based on reinforcement learning and combines Bayesian beliefs over opponent types with PEPPER [8], which in turn builds on R_{MAX} [4]. We follow the assumptions of these state-of-the-art algorithms such as observing local immediate rewards but not the opponents', and knowing the maximum possible reward R_{max} for each episode.

The expected future rewards for a joint action **a** being in state s are defined by:

$$R(s, \mathbf{a}) = r(s, \mathbf{a}) + \sum_{s' \in S} T(s, s', \mathbf{a}) V(s') \tag{1}$$

where $V(s')$ is the expected future rewards of being in state s'. Note that given $r(\cdot), T(\cdot)$ and $V(\cdot)$, value iteration can be used to compute Equation 1. Note that these terms are defined in terms of the joint action, and therefore can be combined with various opponent models to compute expectations and best response policies.

R_{MAX} uses the principle of optimism in face of uncertainty, i.e., initially all states are assumed to result in maximal reward. To learn $r(\cdot)$ and $T(\cdot)$, Pepper uses frequencies of past games and we compute future rewards $V(s)$, by combining off-policy (e.g., Q-learning) and on-policy methods. This is, an on-policy estimation based on the observed distribution of joint actions, using $n(s), n(s, \mathbf{a})$ for the number of visits to state s and the number of times joint action **a** was chosen in that state respectively. Then, the on-policy estimation is given by: $V^{on}(s) = \sum_{\mathbf{a} \in A(s)} \frac{n(s,\mathbf{a})}{n(s)} R(s, \mathbf{a})$ and the combined estimation $V(s) = \lambda(s)\hat{V}(s) + (1 - \lambda(s))V^{on}(s)$. Where $\hat{V}(s)$ represents an optimistic approximation given by $\hat{V}(s) = \max(V^{\text{off}}(s), V^{\text{on}}(s))$ and where $\lambda \in [0, 1]$ represents a stationarity measure initialized to one but approaching zero when the agent gets more experience.[1] We use the concept of non-pseudo stationary restarts [8], i.e., when $R(s)$ is observed to not be pseudo stationary $\lambda(s)$ resets to one. Let $n'(s)$ be the number of visits to stage s since $R(s)$ was last observed to not be pseudo-stationary, then: $\lambda(s) = \max\left(0, \frac{C - n'(s)}{C}\right)$ with $C \in \mathbb{N}^+$.

Algorithm 1: Opponent learning in sequential interactions (OLSI) framework

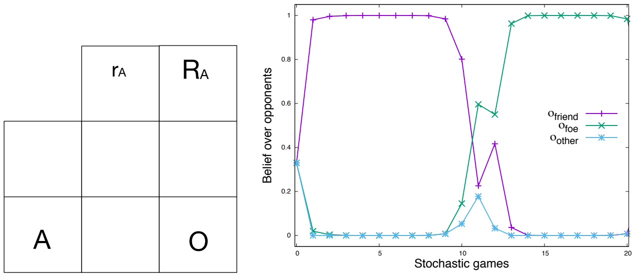
```
Input: Maximum possible reward $R_{max}$, opponent space $\mathcal{H}$, prior probabilities $P(\mathcal{H})$, switch frequency probability $P(s|n)$
1  Initialize beliefs $\beta^0(\mathcal{H}) = P(\mathcal{H})$
2  Initialize $V(\cdot)$ with $R_{max}$
3  Random initial policy $\pi$
4  for each stochastic game $k$ do
5      Update $R(\cdot)$; see Eq. 1
6      Update policy $\pi$ according to $\beta^k(h)$
7      Observe state
8      while state is not goal do
9          Select action $a$
10         Observe state and opponent action $a_{-i}$
11         Receive immediate reward $r$
12         Compute opponent model $\pi_h$ for each opponent $h \in \mathcal{H}$
13         Update belief with observation $a_{-i}$ as $\beta^k(h) = \frac{P(a_{-i}|h)\beta^{k-1}(h)}{\sum_{h' \in H} P(a_{-i}|h')\beta^{k-1}(h')}$
14         if enough visits to $(s, \mathbf{a})$ then
15             Update rewards, $V(\cdot)$ and transition model
16             Update $R(\cdot)$; see Eq. 1
17             Update policy $\pi$ according to $\beta^k(h)$
18     $\beta^{k+1}(h) = P(s|n)\beta^k(h) + (1 - P(s|n))P(\mathcal{H})$
```

Our proposed framework OLSI is described in Algorithm 1. Note that removing the lines that correspond to the belief update (lines 1, 13 and 18) and assuming only one opponent model to compute the policy (lines 6, 12, and 17) yields Pepper. Our framework thus generalizes Pepper by taking into account multiple different hypothesized opponent models (policies) at the same time, computing the agent's best response against the belief over them.

Consider the following common opponent policies: 1) a totally cooperative (friend) opponent, who play its part of maximizing the agent's reward (e.g., due to identical rewards for both): $\pi_{friend} = \max_{\mathbf{a}_{-i}} R(s, \mathbf{a})$, or 2) a competitive (foe) opponent that aims to minimize the agent's reward (e.g., in zero-sum games): $\pi_{foe} = \min_{\mathbf{a}_{-i}} R(s, \mathbf{a})$. Additionally, the agent can 3) learn by observations $\pi_{freq} = \frac{n(s, a_{-i})}{n(s)}$, accounting for mixed types. Note that if the opponent model is correct, it can be used to compute $R(s)$ that accurately reflects future rewards that only depend on the agent's action.

---

[1] Recall that $R(s, a)$ is initialized to R_{max} so it is likely to decrease in early episodes, but eventually will become pseudo-stationary.



(a) (b)

Figure 1: (a) A stochastic game with two players, the learning agent (A) and one opponent (O). The learning agent receives a reward when it reaches state marked with r_A or R_A with $r_A < R_A$. In case of collision the opponent has priority over the state. (b) OLSI's belief over the three opponent types, from games 1-10 OLSI is facing O_{friend}, and then O_{foe} for the remaining games. OLSI's belief accurately represent the opponent identity by being updated at every game and also taking into account the switching frequency (10 in this case).

The belief is updated within a single stochastic game using the observation a_{-i} (opponent action) and the probability of that action being generated by each opponent policy using Bayes' theorem: $\beta^k(h) = \frac{P(a_{-i}|h)\beta^{k-1}(h)}{\sum_{h' \in H} P(a_{-i}|h')\beta^{k-1}(h')}$.

Additionally, OLSI also takes into account the switching frequency of the opponents to update the belief when a game ends. For this, the agent maintains a probability for the event opponent switch, $P(s|n)$, given that n games have been played since the last switch. Then, we update the belief as follows: $P(s|n)\beta^{k-1}(h) + (1 - P(s|n))P(\mathcal{H})$. This is, we consider the probability of facing the same opponent (keeping the same belief) or if a switch is likely to happen then the belief moves to a more conservative prior distribution.

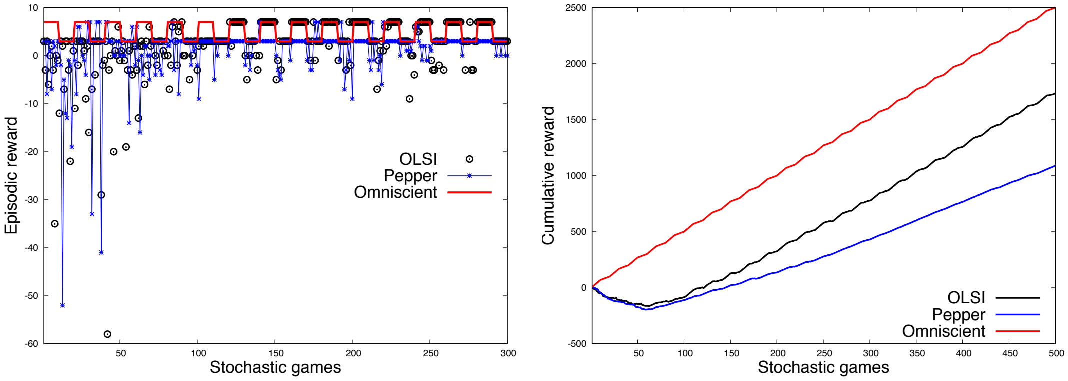
4 Experiments

For the sake of clarity we selected a simple setting with intuitive opponents that simplify the interpretation of the behavior and results. We evaluated our proposed algorithm on a stochastic game represented as a grid world. We performed experiments comparing OLSI, Pepper (with only one opponent model based on frequencies of opponent actions) and an Omniscient agent that knows how to best respond at all times. Figure 1 (a) depicts a graphical representation of the stochastic game used in the experiments. There are two players, the learning agent (A) and the opponent (O). The starting positions are marked with their initial. The learning agent receives a reward when it reaches a goal state r_A or R_A, with $r_a < R_A$. The agents can move upwards or horizontally, and the opponent has the possibility to stay in the same state; the learning agent moves always to the right and the opponent to the left; to avoid agents getting trapped the grid is a toroidal world. With every step that does not transition to a goal state the learning agent receives a penalty p_{min}. In case of collision the learning agent receives high penalty p_{max} with $p_{min} < p_{max}$.

Note that the opponent directly influences the possible reward the learning agent can obtain. For example, since the opponent is closer to R_A it can block its way to the learning agent, in which case the best option would be to go for r_A. For the experiments we set $r_A = 5, R_A = 10, p_{min} = -1, p_{max} = -5$.

We test against a population with two types of opponents, a *foe* opponent, O_{foe}, that aims to block the way to R_A and a *friendly* opponent, O_{friend}, that allows you to pass to R_A. First, we present how the belief behaves in a sequential interaction of 20 stochastic games; during the first 10 games the agent faces O_{friend}, thereafter the agent is paired with O_{foe}. Figure 1 (b) depicts the belief of the OLSI agent, which assumes a opponent space composed of three opponents $\{O_{friend}, O_{foe}, O_{other}\}$, additionally we modelled the switch frequency using a Gaussian distribution $\mathcal{N} = (10, 1)$. OLSI starts with a uniform prior over the three possible opponents and when the first game ends the belief for O_{friend} increases and remains until round 9 where it slightly decreases since the switch frequency is getting closer to the expected value of n. The opponent changes to O_{foe} at game 11, which is reflected in OSLI's belief.

Figure 2 (a) compares episodic rewards of OLSI against Pepper and an Omniscient agent in a sequential interaction of 200 stochastic games. The opponent alternates between O_{friend} and O_{foe} every 10 games. From the figure it can be noticed that approximately in the first 100 games both OLSI and Pepper are learning and obtaining low rewards in some games. However, after this period OLSI learns a best response against each of the two possible opponents obtaining R_A against the friendly opponent and r_A against the foe opponent. In contrast, Pepper learns a safer policy in which in most of the times it opts for the r_A reward even when facing the friendly opponent. Figure 2 (b) depicts cumulative rewards of the compared approaches over 10 repetitions of 500 stochastic games against the same alternating opponents. Results



(a) (b)

Figure 2: (a) Episodic rewards of OLSI, Pepper and the Omniscient agent against the same alternating opponents. Pepper learns to collect r_A as a safe response against the population; in contrast, OLSI best responds by identifying the switching opponents. (b) Corresponding cumulative rewards show OLSI's asymptotic performance parallels the Omniscient's.

show that after the learning phase Pepper obtains lower rewards than the Omniscient agent while OLSI achieves parallel asymptotic performance.

5 Conclusions

This article complements the body of literature on learning algorithms for multiagent systems, which often assumes self-play or stationary opponents, by considering the scenario of repeated stochastic games against switching opponent sets. Our first contribution is the formalisation of this setting of sequential interactions. Our second contribution is a framework, OLSI, for learning in that setting based on reinforcement learning and Bayesian beliefs. OLSI maintains a belief over the hypothesized opponents, which is used to compute an adequate response. Results in a simple scenario showed that OLSI achieved near-optimal results, thus outperforming the conservative strategy learned with a single opponent model. These promising preliminary results suggest to evaluate the effectiveness of OLSI and the effect of parameters (e.g., priors) on algorithm performance in a wider set of experiments with more complex opponents. Once results are validated against further benchmarks, this approach may boost reinforcement learning performance in a broader set of realistic strategic interaction scenarios against changing counter-parties.

References

[1] S. V. Albrecht, J. W. Crandall, and S. Ramamoorthy. Belief and truth in hypothesised behaviours. *Artificial Intelligence*, 235:63–94, 2016.

[2] D. Bloembergen, K. Tuyls, D. Hennes, and M. Kaisers. Evolutionary Dynamics of Multi-Agent Learning: A Survey. *Journal of Artificial Intelligence Research*, 53:659–697, 2015.

[3] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[4] R. I. Brafman and M. Tennenholtz. R-MAX a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

[5] L. Busoniu, R. Babuska, and B. De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[6] D. Chakraborty and P. Stone. Multiagent learning in the presence of memory-bounded agents. *Autonomous Agents and Multi-Agent Systems*, 28(2):182–213, 2013.

[7] V. Conitzer and T. Sandholm. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, 67(1-2):23–43, 2006.

[8] J. W. Crandall. Just add Pepper: extending learning algorithms for repeated matrix games to repeated markov games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 399–406, Valencia, Spain, 2012.

[9] J. W. Crandall. Robust Learning for Repeated Stochastic Games via Meta-Gaming. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, pages 3416–3422, Buenos Aires, Argentina, 2015.

[10] B. C. Da Silva, E. W. Basso, A. L. Bazzan, and P. M. Engel. Dealing with non-stationary environments using context detection. In *Proceedings of the 23rd International Conference on Machine Learnig*, pages 217–224, Pittsburgh, Pennsylvania, 2006.

[11] M. Ghavamzadeh, S. Mannor, J. Pineau, and A. Tamar. Bayesian Reinforcement Learning: A Survey. *Foundations and Trends® in Machine Learning*, 8(5-6):359–483, 2015.

[12] P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research*, 24(1):49–79, 2005.

[13] P. Hernandez-Leal, M. E. Taylor, B. Rosman, L. E. Sucar, and E. Munoz de Cote. Identifying and Tracking Switching, Non-stationary Opponents: a Bayesian Approach. In *Multiagent Interaction without Prior Coordination Workshop at AAAI*, Phoenix, AZ, USA, 2016.

[14] P. Hernandez-Leal, Y. Zhan, M. E. Taylor, L. E. Sucar, and E. Munoz de Cote. Efficiently detecting switches against non-stationary opponents. *Autonomous Agents and Multi-Agent Systems*, 2016.

[15] B. Rosman, M. Hawasly, and S. Ramamoorthy. Bayesian Policy Reuse. *Machine Learning*, 104(1):99–127, 2016.

Artificial Improvisation: Improvisational Theatre with Deep Neural Networks and Robots

Kory W. Mathewson[*]
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
`korym@ualberta.ca`

Piotr Mirowski[†]
HumanMachine
London, UK
`piotr.mirowski@computer.org`

Abstract

This study presents the first report of Artificial Improvisation, or improvisational theatre performed live, on-stage, alongside an artificial intelligence-based improvisational performer. The Artificial Improvisor is a form of artificial conversational agent, or chatbot, focused on open domain dialogue and collaborative narrative generation. Artificial conversational agents are becoming ubiquitous with recent investments fueling research and development at major tech companies. Using state-of-the-art machine learning techniques spanning from natural language processing and speech recognition to reinforcement and deep learning, these chatbots have become more lifelike and harder to discern from humans. Recent work in conversational agents has been focused on goal-directed dialogue focused on closed domains such as trip planning, bank information requests, and movie discussion. Natural human conversations are seldom limited in scope and jump from topic to topic. These conversations are laced with metaphor and subtext. Face-to-face communication is supplemented with non-verbal cues. Live improvised performance takes natural conversation one step further with actors performing in front of an audience. In improvisation the topic of the conversation is often given by the audience during the performance. This suggestion serves to inspire the actors to perform a novel, unique, and engaging scene. During each scene, actors must make rapid fire decisions to collaboratively generate coherent narratives. We have embarked on a journey to perform live improvised comedy alongside artificial intelligence systems. We introduce Pyggy and A.L.Ex. (Artificial Language Experiment), the first two Artificial Improvisors, each with a unique composition and embodiment. This work highlights research and development, successes and failures along the way, celebrates collaborations enabling progress, and presents discussions for future work in the space of artificial improvisation.

Keywords: improvisation, comedy, artificial intelligence, dialogue, chatbot, reinforcement learning, deep neural network, recurrent neural network, natural language processing, text-to-speech

Acknowledgements

The authors would like to thank the generous support and encouragement of Rapid Fire Theatre. This work would not be possible without the countless discussions, support, and motivation of Matt Schuurman, Paul Blinov, Lana Cuthbertson, Julian Faid, Sarah Ormandy, Arthur Simone, Chris Westbury, Jamie Brew, Michael Littman, Tom Sgouros, and Adam Meggido, as well as Alessia Pannese, Stuart Moses, Roisin Rae, John Agapiou, Katy Schutte, Benoist Brucker, Stephen Davidson, Luba Elliot, Shama Rahman, and Steve Roe. For more information, and to see the system in action, visit `http://humanmachine.live`

---

[*] `http://korymathewson.com`

[†] `http://www.cs.nyu.edu/~mirowski/`

1 Introduction

Improvisational theatre, or improv, is the spontaneous creation of unplanned theatrics, often performed live on-stage in front of an audience. Improv is a form of collaborative interactive storytelling, where two or more people work together to generate novel narratives. It is grounded in the connections between the performer(s) and the audience. Improv requires the performers to work as a team. The actors must rapidly adapt, empathize, and connect with each other to achieve natural, fluid collaboration. To truly excel at the art form, performers must think and react to audiences reactions quickly, and work together to accept and amplify each other's offers—an act that can be seen as *real-time dynamic problem solving* [1]. Improv demands human performers handle novel subject matter through multiple perspectives ensuring the audience is engaged while progressing narrative and story. Due to the incredible difficulty improvisors must embrace failure quickly and gracefully and give in to spontaneity [2].

Computer aided interactive storytelling has been considered in previous work mainly for applications in video games [3]. This work aims to create endless narrative possibilities in video game universes thereby maximizing user engagement. Robotic performances have been explored previously. In 2000, Tom Sgorous performed *Judy, or What is it Like to Be A Robot?* [1]. In 2010, the realistic humanoid robot Gemenoid F performed *Sayonara* [2], which was later turned into a movie [3]. In 2014, Carnegie Mellon University's Personal Robotics Lab collaborated with their School of Drama to produce *Sure Thing* [4] [4]. In these performances, the robots were precisely choreographed, deterministic or piloted on stage. These shows required the audience to suspend their disbelief and embrace the mirage of autonomy. These performances verge ever closer to the deep cliffs surrounding the uncanny valley—the idea that as the appearance of a humanlike robot approaches, but fails to attain, human likeness, a person's response would abruptly shift from empathy to revulsion [5].

Our work serves as a bridge between the artificial intelligence labs and improvisational theatre stages. We aim to build a bridge over the uncanny valley, toward a future where humans and autonomous agents converse naturally. Our work is partially inspired by the narratives behind George Bernad Shaw's Pygmalion [6], Mary Shelly's Frankenstein [7], and Alan Jay Lerner's My Fair Lady [8]. In these stories, creators attempt to design and build reflections of themselves, fabricating their respective ideal images of perfection.

We present our methods and details on the systems which power the first two Artificial Improvisors. We concisely report on findings, and discuss future work at the intersection between artificial intelligence and improvisational theatre.

2 Methods

Over the last year we iterated from *Version 1: Pyggy* using classic machine learning and deterministic rules to *Version 2: A.L.Ex.* which uses deep neural networks, advanced natural language processing, and a much larger training dataset. An Artificial Improvisor is composed of three major building blocks: 1) speech recognition, 2) speech generation and 3) a dialogue management system. The three modules comprise a simplified framework, inspired by the *General Architecture of Spoken Dialogue Systems*, for extemporaneous dialog systems [9]. We will present details on these components for both Pyggy and A.L.Ex.

2.1 Version 1: Pyggy

Pyggy, short for Pygmalion, was an early version of an Artificial Improvisor [5]. Pyggy was built using speech recognition powered by Google Cloud Speech API [6]. This speech recognition translated sound waves from human voice, to text through a network-dependent API. Pyggy used Apple Speech Synthesis for translated output text to sound. Dialogue management was handled with Pandorabots [7] and Chatterbot [8].

Pandorabots handled hard-coded rules and deterministic responses. For example, when the human said: "Let's start improvising", the system would *always* respond: "Ok". Pandorabots also handled saving simple named entities. For example, if the human said: "My name is Kory" then the system could answer the question: "What is my name?". Chatterbot handled more open dialogue and added a sense of randomness to the system. Chatterbot was pre-trained on a set of dialogue and then learned based on responses the human gave back to the system. For a given human improvisor statement, each of these systems would generate their respective response, and the responses were concatenated and then output back to the user.

Pre-training of Pyggy was done through an interactive website where individuals could directly interact. Unfortunately, when the general public had the ability to interact with Pyggy many started to act adversarially and mischievously,

---

[1] `http://sgouros.com/judy/`

[2] `http://www.geminoid.jp/en/art.html`

[3] `https://wikipedia.org/wiki/Sayonara_(2015_film)`

[4] `http://www.cs.cmu.edu/~garthz/research/HERB-drama/drama.html`

[5] `http://korymathewson.com/building-an-artificial-improvisor/`

[6] `https://cloud.google.com/speech/`

[7] `http://www.pandorabots.com/`

[8] `https://github.com/gunthercox/ChatterBot`

training the system to say rude and inappropriate things. Once this training set was cleaned and filtered, it was quite small. Additional clean training data was appended from the Cornell Movie Dialogue Corpus [10]. The dataset is composed of 220579 conversational exchanges from 617 movies and provided the system plenty of novel, interesting, unique dialogue to pull from.

Pyggy is embodied by a visualization as seen in Fig. 1. The visualization of Pyggy was accomplished with Magic Music Visualizer [9]. The simple animation system shifted the mouth slightly when sound was being played by the speech generation system. This effect gave Pyggy an animated life and a face on stage.

2.2 Version 2: A.L.Ex. (Artificial Language Experiment)

There were limitations to the dialogue which Pyggy could produce, as it was restricted to the set of training data. The very nature of improvised theatre is novel generative conversational abilities. While building an open-domain conversational agent is AI-hard, a generative dialogue system that is conditioned on previous text and that mimics collaborative writing (as in the Surrealists' *Cadavres Exquis*) could give to the audience an illusion of sensible dialogue [11]. The need for generative dialogue required shifting from the rule-based, deterministic learning systems of Pyggy to deep neural network-based dialogue management. This was done by implementing a statistical language model which would generate sentences word by word. [10]. This new model, called A.L.Ex (Artificial Language Experiment) was built using recurrent neural networks (RNN) with long-short term memory (LSTM) [12, 13]. Full details of the architecture are excluded from the current study.

Furthermore, A.L.Ex. was designed to subvert the multiplicity of connected services which formed the architecture of Pyggy. A.L.Ex. aimed to be an offline, standalone Artificial Improvisor. While, similarly to Pyggy, speech recognition and generation are still performed by ready-made tools, respectively Apple Enhanced Dictation [11] and Apple Speech Synthesis, these tools are run on the same computer. Moreover, the entire text-based dialogue system (coded in Lua and Torch by Piotr Mirowski and trained on transcribed subtitles from 100k movies [12]), was encapsulated into a single program which then makes system calls to speech recognition and text-to-speech, and was controlled through a graphical user interface which visualizes results (both the recognized and generated sentences in the dialogue). The core system could be extended with additional modules, as it also ran a fault-resilient server that accepts incoming HTTP GET requests from client applications. These applications include software controlling a humanoid robot with pre-programmed motions that are activated when A.L.Ex speaks (see Figure 2). Applications have been written for controlling both the EZ-Robot JD Humanoid [13] and the Aldebaran Nao [14].

3 Results

There are challenges associated with testing, and quantitatively evaluating, open-domain dialogue systems systems [14, 11]. One reasonable measure for qualitative assessment is the audience perceived performance level of an Artificial Improvisor as an improvisational performer. Thus, each of these systems has been tested several times live in front of an audience. Through audience feedback, the system has been iteratively improved.

3.1 Version 1: Pyggy

Pyggy performed in the show *Artificial Improvisation* live in-front of an audience on April 8, 2016. The performance, an invited performance during Rapid Fire Theatre's 2016 Bonfire Festival[15], was well received by the audience. Here is a short excerpt from the performance:

KORY: We're having a picnic! Would you like cheese and bread?

PYGGY: Seen one human, you've seen them all.

KORY: Promise me, right now, we're here listening to each other.

PYGGY: For fun? Where are all your queer party friends?

KORY: Why don't you have any feelings for me? That's why I had to invite you on a picnic. I wanted to get some time to talk to each other.

PYGGY: I miss your vivid sense of humour.

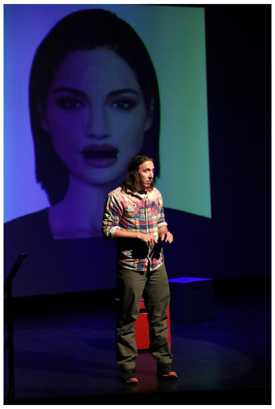


Figure 1: Kory performing on-stage with Pyggy.

---

[9] https://magicmusicvisuals.com/

[10] https://goo.gl/D4c5OX

[11] https://support.apple.com/en-ca/HT202584

[12] OpenSubtitles https://www.opensubtitles.org and http://opus.lingfil.uu.se/OpenSubtitles2016.php

[13] Manufactured by EZ-Robot https://www.ez-robots.com

[14] Manufactured by SoftBank Robotics https://www.ald.softbankrobotics.com/en

[15] http://rapidfiretheatre.com/festival/bonfire/

3.2 Version 2: A.L.Ex. (Artificial Language Experiment)

A.L.Ex. performed for the first time in front of a live audience on July 24, 2016, and has performed over fifteen times and on two continents since then. Here is a short transcript from an early performance with A.L.Ex.:

ALBERT: Captain, the ship is under attack. The frigate is arriving...

A.L.EX: You see, I don't feel the same way. I just don't want to stay in the position to get burnt.

ALBERT: We're all going to get burned by a cannon fire...

A.L.EX: While you're in there, tell me. When I get dressed, I'll go up there and see what happens.

ALBERT: Captain, you need to get on the deck, the crew is expecting you to speak, to take leadership. We are in the middle of a war and this is an invasion of foreign ships. Captain, show us your leadership.

A.L.EX: We won't be able to navigate without them as we are, serving for us to act like a life of violence and to help us die from the most powerful life of our time.

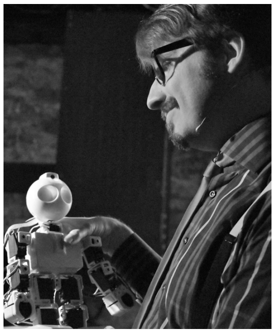


Figure 2: Albert (Piotr Mirowski) performing onstage with A.L.Ex. Image credits: Robert Stafford.

4 Discussion

Future work will incorporate recent advances in deep reinforcement learning for dialogue generation. Through design of reward functions, more interesting dialogue may be encouraged. Three useful conversational properties recently shown to improve long-term success of dialogue training are: informativity, coherence, and ease of answering [15]. Additional reward schemes may improve, or tune, the trained deep neural network based dialogue managers. Recent work has shown that reinforcement learning can be used to tune music generation architectures [16]. Rewarding linguistic features (i.e. humor, novelty, alliteration) may prove useful in the generation of creative and interesting statements [17].

Natural human conversations are seldom limited in scope, jump from topic to topic, and are laced with metaphor and subtext. Artificial Improvisors of the future should make use of recent advances in artificial memory and attention models. As well, humans often make use of non-verbal cues during dialog. By incorporating this additional information, human(s) could both consciously and subconsciously inform the learning system [18, 19]. Additionally, if the Artificial Improvisor is modeled as a goal-seeking agent, then shared agency be quantified and communicative capacity could be learned and optimized during the performance [20].

While the system is trained to perform dialog, it is not trained to tell a cohesive story with a narrative arc. The addition of memory network advancements may improve callback, additional engineering and training will be necessary to collaboratively build a narrative arc. In 1928, William Cook published a book on algorithmic plot development which may serve this purpose, and implementations and connections have yet to be explored [21].

There is a large research area devoted to computers and creativity [22]. The field of artificial improvisation is growing rapidly, fueled by recent advances in natural language processing, speech, and music generation [23, 24]. These advances are pushed to the horizon everyday by artists around the world. These artists build complex projects, and use state-of-the-art machine learning techniques to implementing creative, imaginative ideas. On the forefront of this are creative technologists, including Ross Goodwin [16] and Jamie Brew [17].

Thought must be given to the interface through which the human and artificial performers interact [25, 26]. The embodiment of the Artificial Improvisor has been investigated with Pyggy and A.L.Ex. using on-screen visualizations and robotics. Stage presence is critical to ensure that an Artificial Improvisation show is watchable, enjoyable, and engaging. Improvisational performances are not strictly conversational narrative performances and often demand intense physicality from the performers. The optimal means by which these systems take the stage has yet to be determined. Collaboration between scientists and creatives will lead to innovative interactions, and immersive art which ignites the senses. With the growing popularity of interactive mixed-reality experiences (e.g. virtual and augmented reality), and the multi-billion dollar video game industry, focus continue to be cast on improving artificial conversational agents and interactive creative dialogue generation.

Improvisational theatre is a domain where experimentation is encouraged, where interaction is paramount, and where failure flourishes. It allows artificial intelligence agents to be effectively tested, and audience reaction can provide a subjective measure of improvement and cognizance. While this may feel similar to the Turing Test, an early attempt to

---

[16] `http://rossgoodwin.com/`

[17] `http://jamiebrew.com/`

separate mind from machine through a game of imitation, deception and fraud, we believe it is much more [27]. Success will be measured by audience preference to engage in shows incorporating Artificial Improvisation and human desire to participate in these dialogues.

Games, like chess and Go, are complex but they are computationally tractable for computers and solutions can be approximated. Improvisational theatre demands creativity, rapid artistic generation, and natural language processing. Improv is not a zero-sum games especially as these systems learn to converse open-domain settings [14, 11].

Future work will continue to explore the evaluation of performance in such an open domain. Performances with Artificial Improvisors continue to spur questions and insights from other performers and audiences alike. We look forward to fulfilling the distant goal of the human observer as a fly on the wall watching AI-performers on-stage improvising in front of an audience full of AI-observers :) In that process, we will engage in a constructive dialogue about creative A.I. aiming at building bridges between the sciences and the arts.

References

[1] B. Magerko *et al.*, "An empirical study of cognition and theatrical improvisation.," in *Creativity and Cognition*, 2009.

[2] K. Johnstone, *Impro. Improvisation and the theatre.* Faber and Faber Ltd, 1979.

[3] M. O. Riedl and A. Stern, "Believable agents and intelligent story adaptation for interactive storytelling," in *International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, pp. 1–12, Springer, 2006.

[4] G. Zeglin *et al.*, "Herb's sure thing: A rapid drama system for rehearsing and performing live robot theater," in *2014 IEEE International Workshop on Advanced Robotics and its Social Impacts*, pp. 129–136, Sept 2014.

[5] M. Mori, K. F. MacDorman, and N. Kageki, "The uncanny valley [from the field]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 98–100, 2012.

[6] G. B. Shaw, *Pygmalion*. Simon and Schuster, 2012.

[7] M. W. Shelley, *Frankenstein: The 1818 Text*. WW Norton, 1996.

[8] G. Cukor, "My fair lady," 1964.

[9] O. Pietquin, *A framework for unsupervised learning of dialogue strategies*. Presses univ. de Louvain, 2005.

[10] C. Danescu-Niculescu-Mizil and L. Lee, "Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs.," in *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics, ACL 2011*, 2011.

[11] R. Higashinaka *et al.*, "Towards an open-domain conversational system fully based on natural language processing.," in *COLING*, pp. 928–939, 2014.

[12] T. Mikolov *et al.*, "Recurrent neural network based language model.," in *Interspeech*, vol. 2, p. 3, 2010.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] J. Glass, "Challenges for spoken dialogue systems," in *Proceedings of the 1999 IEEE ASRU Workshop*, 1999.

[15] J. Li *et al.*, "Deep reinforcement learning for dialogue generation," *arXiv preprint arXiv:1606.01541*, 2016.

[16] N. Jaques *et al.*, "Tuning recurrent neural networks with reinforcement learning," *CoRR*, vol. abs/1611.02796, 2016.

[17] G. Hollis, C. Westbury, and L. Lefsrud, "Extrapolating human judgments from skip-gram vector representations of word meaning," *The Quarterly Journal of Experimental Psychology*, pp. 1–17, 2016.

[18] K. W. Mathewson and P. M. Pilarski, "Simultaneous Control and Human Feedback in the Training of a Robotic Agent with Actor-Critic Reinforcement Learning," in *IJCAI Interactive Machine Learning Workshop*, 2016.

[19] K. W. Mathewson and P. M. Pilarski, "Actor-critic reinforcement learning with simultaneous human control and feedback," *Under review for the 34th International Conference on Machine Learning. Sydney, Australia*, 2017.

[20] R. S. S. Pilarski, Patrick M. and K. W. Mathewson, "Prosthetic devices as goal-seeking agents," in *Second Workshop on Present and Future of Non-Invasive Peripheral-Nervous-System Machine Interfaces, Singapore*, 2015.

[21] W. W. Cook, *Plotto: a new method of plot suggestion for writers of creative fiction*. Ellis., 1928.

[22] J. McCormack and M. dInverno, "Computers and creativity: The road ahead," in *Computers and creativity*, pp. 421–424, Springer, 2012.

[23] A. v. d. Oord *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[24] S. O. Arik *et al.*, "Deep Voice: Real-time Neural Text-to-Speech," *ArXiv e-prints*, Feb. 2017.

[25] M. Yee-King and M. dInverno, "Experience driven design of creative systems," in *Title: Proceedings of the 7th Computational Creativity Conference (ICCC 2016). Universite Pierre et Marie Curie*, 2016.

[26] J. McCormack and M. dInverno, "Designing improvisational interfaces," 2016.

[27] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

Shared Learning in Ensemble Deep Q-Networks

Rakesh R Menon[*]
IIT Madras
ee13b056@ee.iitm.ac.in

Manu Srinath Halvagal[*]
IIT Madras
ee13b119@ee.iitm.ac.in

Balaraman Ravindran
IIT Madras
ravi@cse.iitm.ac.in

Abstract

Most deep RL solutions still use extensions of conventional exploration strategies that have been well studied and offer theoretical guarantees in bandit problems and simple MDPs. However, exploration in large state spaces needs to be more directed than is possible with these traditional exploration strategies such as ϵ-greedy. The recently proposed Bootstrapped DQN [4] offers a new exploration strategy that is capable of deep directed exploration and is better suited for deep RL problems.

Bootstrapped DQN works by learning multiple independent estimates of the action-value function and guiding action selection using a randomly selected estimate. The method relies on variability among the different value estimators (called heads) for effective and deep exploration. In [4], this variability is ensured through both selective masking of training examples as well as by the random initialization of network parameters of each head. The network is trained using the Double DQN [5] update rule. Double DQN is an adaptation of Double Q-Learning [2] which is meant to reduce the overestimation bias in Q-learning [6].

In both Double DQN and Bootstrapped DQN, the target network is used as a stand-in for an independent estimate of the action-value function in the update rule. Independent estimates are needed for Double Q-learning to perform effective updates. However, the target network is highly coupled to the online network leading to imperfect double Q-learning updates.

We propose *shared learning*, an algorithm which takes advantage of the ensemble architecture of Bootstrapped DQN to overcome the issue with coupled estimates described above. Further, we supplement our algorithm with a framework to share learned experience amongst the bootstrapped heads. We demonstrate how this method can help in speeding up the existing Bootstrapped DQN algorithm with minimal computational overhead.

Keywords: Reinforcement Learning, Double Q-learning, Deep Q-Networks, Ensemble Networks

---

[*] Equal Contribution

1 Introduction

Although the exploration-exploitation dilemma is well-studied in reinforcement learning problems, it is still a major challenge. This is because existing techniques do not scale well to environments with large state spaces. Bootstrapped DQN introduces a novel exploration strategy based on ensembles of value estimates that address these scalability issues to a large extent. However the estimates of the target networks in Bootstrapped DQN are closely coupled to the online networks, which leads to imperfect double Q-learning updates. While this problem existed in the original work on Double DQN, solving it was not straightforward without the ensemble architecture available in Bootstrapped DQN.

In this work, we propose *shared learning*, which reduces this coupling and generates more robust target estimates. We show the efficacy of our algorithm on the Atari 2600 domain as compared to Bootstrapped DQN. We also analyze the benefits of sharing experience through experiments on toy MDP chains and the puddle world environment.

2 Background

2.1 Double Deep Q-Networks

The max operator in the Q-learning[6] update has been shown to produce overestimation of the value function in [2]. This comes about because a single estimator is used both for choosing the next action and for giving an estimate for the value of that action. Double Q-learning [2] reduces the overestimations by decoupling the action selection and value estimation by training two estimators $Q^A(s,a)$ and $Q^B(s,a)$. The update rule for double Q-learning is given as:

$$Q^A_{t+1}(s_t,a_t) = Q^A_t(s_t,a_t) + \alpha(r_t + \gamma Q^B_t(s_{t+1}, \text{argmax}_a Q^A_t(s_{t+1},a)) - Q^A_t(s_t,a_t)) \tag{1}$$

[5] extends this to DQN[3] in a computationally efficient manner by taking advantage of the target network. The greedy policy is evaluated using the online network and the value is given by the target network for this update. Thus the loss signal for the Double DQN (DDQN) is given as:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma Q(s', \text{argmax}_{a'} Q(s',a';\theta_i); \theta_i^-) - Q(s,a;\theta_i))^2] \tag{2}$$

2.2 Bootstrapped DQN

Bootstrapped DQN [4] introduces a novel exploration strategy that is capable of performing deep exploration in large state spaces. The key idea behind Bootstrapped DQN is the use of randomized value function estimates to approximate a distribution over possible action-values. At the start of every episode, Bootstrapped DQN samples a single value function estimate at random according to a uniform distribution. The agent then follows the greedy policy with respect to the selected estimate until the end of the episode. The authors propose that this is an adaptation of the Thompson sampling heuristic to RL that allows for temporally extended (or deep) exploration. Bootstrapped DQN is implemented by adding multiple head networks which branch out from the output of the CNN as shown in Figure 1. Suppose there are K heads in this network. The outputs from each head represent different independent estimates of the action-value function. Let $Q_k(s,a;\theta_i)$ be the value estimate and $Q_k(s,a;\theta_i^-)$ be the target value estimate of the kth head. The loss signal for the kth head is exactly that given in equation 2. Each of the K heads is updated this way and the gradients are aggregated and normalized at the CNN layers.

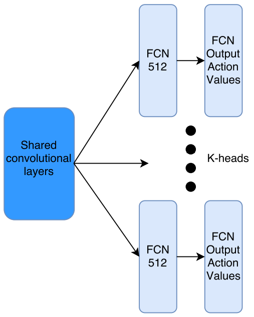


Figure 1: Bootstrapped DQN architecture

3 Shared Learning

3.1 Coupled Estimates in Double DQN

Bootstrapped DQN uses double Q-learning for updating its estimates of the value function. Following the Double DQN algorithm, the value for the update for each head is given by the target network corresponding to the same head. As shown in equation 2, the greedy action for the target value is chosen by the online network. This, however, poses an issue because the target network is a previous iteration of the online network, which effects a close coupling between them. The coupling reduces the effectiveness of the double Q-learning update. This problem has been identified in [5] as well.

3.2 Proposed Algorithm

The ensemble architecture put forward in [4] suggests a computationally efficient way to circumvent the problem described above. To this end, we propose an algorithm that generates decoupled target value estimates by taking advantage of the fact that the each head in Bootstrapped DQN already maintains a value estimate that is completely independent of the other heads.

Consider the K-headed Bootstrapped Deep Q-Network described in Section 2.2. In order to decouple the estimates, we use the values from a head other than the one being updated, to pick the greedy action for the target value in the double Q-learning update. The new loss signal can be formulated as follows:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma Q_k(s', \text{argmax}_{a'} Q_m(s', a'; \theta_i); \theta_i^-) - Q_k(s, a; \theta_i))^2] \tag{3}$$

where $m \in \{1, 2, \ldots, K\} \setminus \{k\}$.
The target value given here is expected to be a more robust estimate when compared to Double DQN, since it is derived from a completely independent estimate of the value function.

3.2.1 Online Transfer of Learning Progress

The head selected for picking the greedy action for the target value in equation 3 provides a degree of guidance to the head being updated, allowing transfer of some amount of learned knowledge. Hence, basing the selection of this head on a measure of local learning progress rather than choosing an arbitrary head seems a better choice. This would be in some sense an online transfer of learned knowledge. More concretely, at any point of time, we would expect at least one head to have learned more about the current region of the state space than the other heads. Online transfer of experience from this head to the rest of the ensemble network could aid learning in the other heads and result in more directed exploration. With this in mind, we propose the *shared learning* algorithm which modifies the loss signal from equation 3 in the following way:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma Q_k(s', \text{argmax}_{a'} Q_{best}(s', a'; \theta_i); \theta_i^-) \\ - Q_k(s, a; \theta_i))^2] \tag{4}$$

Here $Q_{best}(s', a'; \theta_i)$ represents the action-values of the 'best' head (selected at regular intervals) that has progressed furthest in learning about the current state. In our implementation, we quantify the learning progress using the action-values of the online network. This measure was chosen because the head which outputs the highest action-value is expected to have learned off of more rewarding subsequent states. The complete algorithm has been summarised in Algorithm 1.

Algorithm 1 Shared Learning Algorithm

```
Input: Value function networks $Q$ with $K$ outputs $\{Q_k\}_{k=1}^K$
Let B be a replay buffer storing experience for training and select_best_int be the interval during which the best head is selected.
numSteps ← 0
best_head ~ Uniform$\{1, 2, \ldots, K\}$
for each episode do
    Obtain initial state from environment $s_0$
    Pick value function to act using $k$ ~Uniform$\{1, 2, \ldots, K\}$
    for step $t = 1, \ldots$ until end of episode do
        Pick an action according to $a_t \in \text{argmax}_a Q_k(s_t, a)$
        Take action $a_t$ and receive state $s_{t+1}$ and reward $r_t$
        Sample minibatch from B
        Train network with loss function given in equation 4
        numSteps ← numSteps + 1
        if numSteps $mod$ select_best_int == 0 then
            best_head = $\text{argmax}_k Q_k(s_t, a_t)$
        end if
    end for
end for
```

4 Experiments

We first illustrate how online transfer of learning progress can speed up learning on an MDP chain. In this tabular environment, the issue of coupling between target estimates and online estimates is nonexistent, and any speedup due to our algorithm would primarily be because of online transfer. Finally, we demonstrate the effectiveness of our algorithm on a subset of Atari Games using the ALE environment.

4.1 MDP

We present a toy chain MDP experiment to demonstrate the faster learning of the new approach. The MDP consists of n states, arranged sequentially as shown in Figure 2. The agent always starts from state s_2. From each state *four* actions are allowed namely, go left, go right, do nothing or jump to state s_1 incurring a reward of -10 and terminating the episode. The episode ends when the agent has reached the nth state upon which it receives a reward of +10. We use normal Q-learning (with ϵ-greedy) and double Q-learning (with ϵ-greedy) as baselines to compare the performance of a bootstrap agent (5 heads) with and without shared learning. Figures 3a and 3b show the result for a 35 and 50 state MDP respectively.

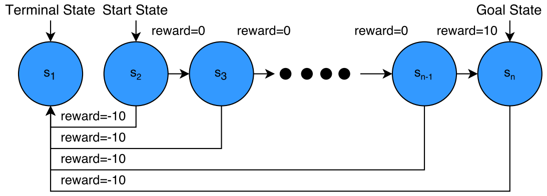
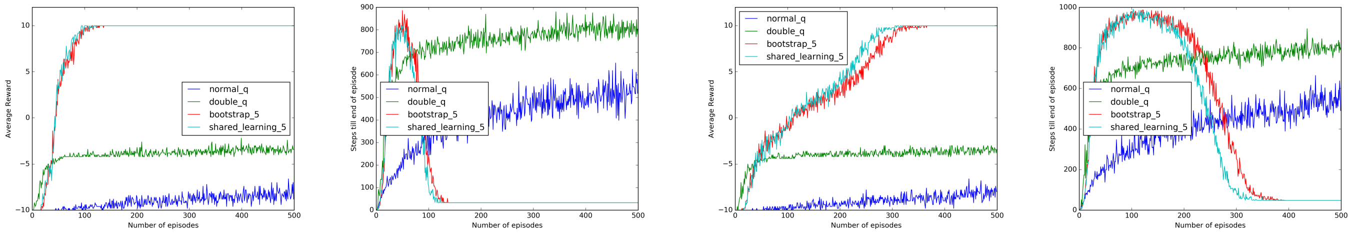


Figure 2: MDP Chain Structure with n-states



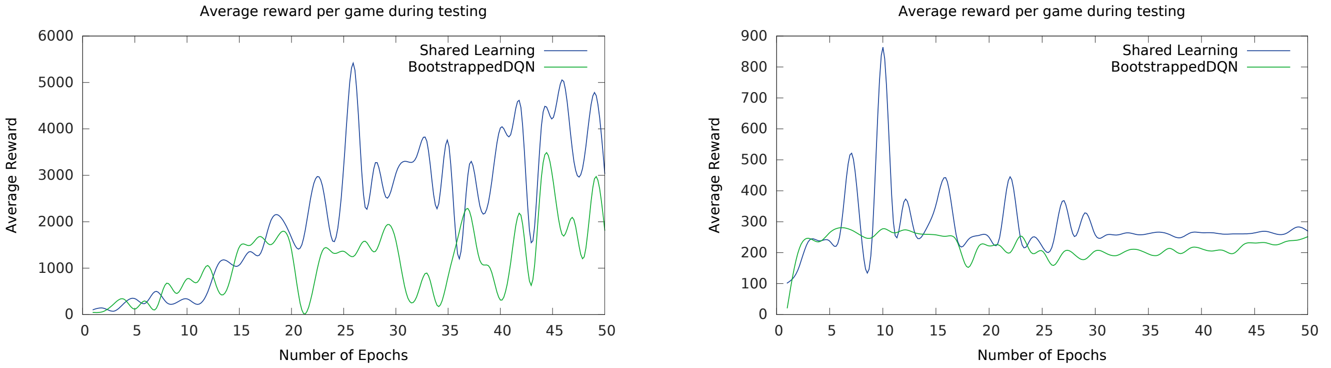
(a) 35-state chain MDP (b) 50-state chain MDP

Figure 3: MDP results showing average reward obtained per episode and number of steps taken until completion

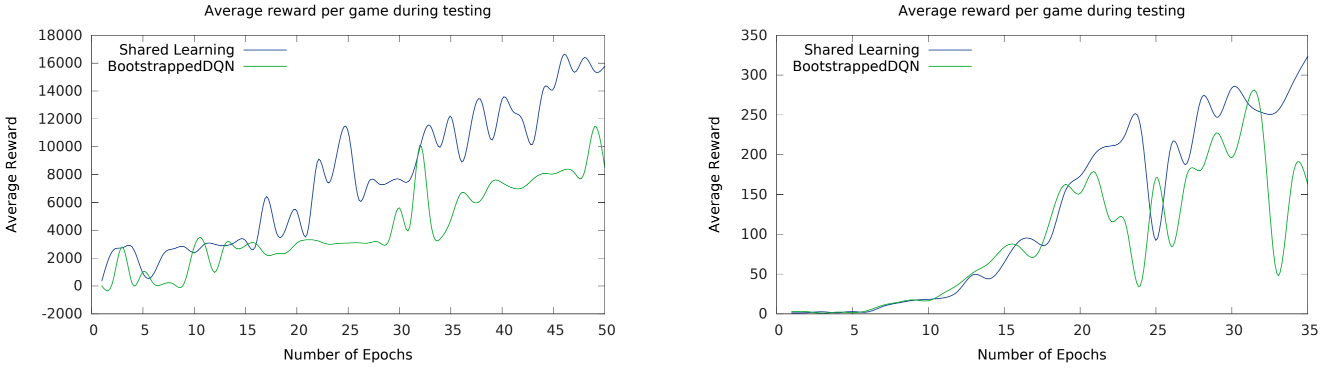
Solving the above environment requires deep exploration, especially with a larger number of states. This is illustrated by the fact that Q-learning and double Q-learning, with ϵ-greedy exploration, are unable to solve large MDP chains (beyond 20 states). The results verify that Bootstrapped DQN (adapted for tabular settings) can indeed perform deep exploration. The fact that our algorithm is still capable of performing deep exploration shows that sharing does not take away from the diversity among the estimates, which is what drives the exploration in Bootstrapped DQN.

We observe that the speedup due to shared learning becomes more evident with increasing chain length. The performance of both Bootstrapped DQN and *shared learning* would probably be identical on this problem until the first time the goal state is reached. It is at this point that sharing learned experience becomes vital so that knowledge of the goal state propagates out to every head. This is the reason why *shared learning* outperforms the bootstrap algorithm on larger MDP chains.

4.2 Arcade Learning Environment



(a) Average Reward on Seaquest after 50 epochs of training (b) Average Reward on Frostbite after 50 epochs of training



(c) Average Reward on Hero after 50 epochs of training (d) Average Reward on Breakout after 35 epochs of training

Figure 4: Results on Arcade Learning Environment

We evaluate our algorithm on 6 Atari games on the Arcade Learning Environment [1]. The games chosen for evaluation were Seaquest, Frostbite, Hero, Breakout, Qbert and Pong. Most of these games have been chosen in order to compare with the results shown in [4]. The network architecture used for shared learning is the same as that of Bootstrapped DQN and consists of a shared convolutional layer followed by 10 bootstrap heads to provide value function estimates. Gradient normalisation of $1/K$ (K is the number of bootstrap heads) was also applied to the network with no masking of heads. The learning progress for the Atari games have been measured using the Q-values at intervals of 100 steps. The head corresponding to the maximum Q-value is chosen to be the best head that is suitable for information transfer. In Figures 4a, 4b and 4c, we show the results on Seaquest, Frostbite and Hero for Shared Learning against Bootstrapped DQN over 50 epochs and for Breakout in Figure 4d over 35 epochs.Table 1 shows the cumulative rewards obtained on the Atari games Qbert and Pong over 20 epochs.

Cumulative Reward on Atari Games during testing		
Game	Shared Learning Score	Bootstrapped DQN Score
Qbert	**39001.6661**	15286.0259
Pong	**-44.9735**	-80.4237

Table 1: 20 epoch cumulative scores

We have not presented the performance of DQN [3] and double DQN [5], since it was shown in [4] that Bootstrapped DQN clearly outperforms them. Our results show an increased cumulative reward during learning, while Bootstrapped DQN appears to dither to a larger extent. The significant boost in performance on the Atari games over Bootstrapped DQN confirms that the double Q-learning updates in DDQN and Bootstrapped DQN were in fact imperfect.

5 Conclusion and Future Work

In this work, we present the *shared learning* algorithm which performs learning updates closer to the true double Q-learning rule than Bootstrapped DQN and Double DQN. We achieve this without affecting the deep exploration capabilities of Bootstrapped DQN. The algorithm involves minimal computation over the already efficient framework of Bootstrapped DQN.

We also introduce the notion of online transfer of knowledge for more directed exploration. This is more evident in the results of the MDP chain experiments. It can also be observed to a lesser extent on the Atari games. While we have studied shared transfer within the Bootstrap framework, online transfer can be integrated into any ensemble learning method for RL. Additionally, we would like to test this idea of online transfer on a wider array of AI tasks.

An important issue that needs to be studied further is the criterion for picking the best value estimator. Saliency-based measures might make for better metrics in environments with sparse rewards where promoting exploration is more beneficial than maximizing rewards.

References

[1] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

[2] Hado V Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015.

[4] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep Exploration via Bootstrapped DQN. In *Advances In Neural Information Processing Systems*, pages 4026–4034, 2016.

[5] Hado Van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-Learning. In *AAAI*, pages 2094–2100, 2016.

[6] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Aging of the Exploring Mind: Older Adults Deviate more from Optimality in Complex Environments

Job J. Schepens[*]
Center for Cognitive Neuroscience Berlin
Freie Universität
Berlin 14195, Germany
job.schepens@fu-berlin.de

Ralph Hertwig
Center for Adaptive Rationality
Max Planck Institute for Human Development
Berlin 14195, Germany
hertwig@mpib-berlin.mpg.de

Wouter van den Bos
Center for Adaptive Rationality
Max Planck Institute for Human Development
Berlin 14195, Germany
vandenbos@mpib-berlin.mpg.de

Abstract

Older adults (OA) need to make many important and difficult decisions. Often, there are too many options available to explore exhaustively, creating the ubiquitous tradeoff between exploration and exploitation. How do OA make these complex tradeoffs? OA may generally more often rely on model-free than model-based reinforcement learning. Such computational changes have been associated with age-related changes in cortico-striatal control loop functioning. Here, we investigated age-related shifts in solving exploration-exploitation tradeoffs depending on the complexity of the choice environment. Older (N = 32, mean age = 70.47) and younger adults (N = 29, mean age = 24.31) played four and eight option N-armed bandit problems with the numbers of current gambles and average rewards displayed on the screen. This minimized the role of working memory and allowed us to focus on how OA learn to seek knowledge effectively. This is a relevant issue in studies of continually learning artificial agents as well. OA reliably chose the most rewarding options less often than younger adults (YA) did. In addition, choices of OA were more deviant from an optimality model (Thompson sampling). This optimality model tracks uncertainty beyond simple action values or only the last choice. We further measured structural connectivity in cortico-striatal loops using diffusion weighted MRI (pending analyses). Together, OA seem to process uncertainty that is associated with options in more complex choice environments sub-optimally, suggesting more limited task representations. This interpretation fits to multiple contexts in the complex cognitive aging literature, and in particular to the context of challenges in the maintenance of goal-directed learning. Such changes may result from constraints of biological aging as well as the cognitive processes of continuous information selection and abstraction needed for building new knowledge on existing knowledge over the life-span.

Keywords: Cognitive aging, learning mechanism, complex choice environment, N-armed bandits, optimality model

Acknowledgements

European Commission Horizon 2020 MSCA No 708507. Robert Lorenz, Jann Wscher, Dania Esch, and Lukas Nagel supported with participant recruitment, organization, and data collection.

---

[*] Elicitation materials, data, and analysis scripts can be found on github.com/jobschepens/Bandits/

1 Introduction

In today's aging societies, more and more older adults (OA) are making cognitively demanding decisions about work, finances, their health, etc. Many such decisions benefit from thinking about future goals because the options available create explore-exploit tradeoffs. How do OA usually respond to these cognitive challenges in increasingly complex choice environments? Decision makers generally have access to a number of learning mechanisms, including Pavlovian reflex based learning, habitual experience-based learning, and goal-directed learning. Goal-directed learning depends on some internal model, so that learning can be adapted flexibly, like when e.g. managing a research project. Habitual learning has been related to a dorsolateral striatal to sensorimotor cortex control loop while goal-directed learning has been related to a dorsomedial striatal to ventromedial and lateral prefrontal cortex control loop (Daw & O'Doherty, 2014). Importantly, goal-directed learning is impaired in OA and this impairment has been associated to lower activation in prefrontal cortex areas (Eppinger & Bruckner, 2015; Eppinger, Walter, Heekeren, & Li, 2013). OA rely relatively more often on experience-based learning, which may arise from white matter integrity changes in the ventromedial and lateral prefrontal cortex (Chowdhury et al., 2013; Eppinger et al., 2013; Samanez-Larkin & Knutson, 2015).

It is currently unclear how learning mechanisms in OA depend on the relative complexity of a task. Such a dependency is likely because of the adaptive fit between the mind and the environment, following an ecological rationality perspective (Mata et al., 2012). This perspective predicts that simpler strategies will be favored, depending on what strategy an environment affords (Mata, Schooler, & Rieskamp, 2007). For example, when OA need to explore options in order to choose between them later, OA rely on more minimal exploration strategies than YA in more complex environments (Frey, Mata, & Hertwig, 2015). Here, we study such age-related performance changes in explore-exploit tradeoffs with varying cognitive demands. Analyzing effects of the complexity of choice environments this way could help identify age-related differences in learning mechanisms.

We used typical N-armed bandit problems to study changes in learning mechanisms across choice environments. Participants made inferences about risky options by sampling information from a four and eight option choice environment. Rewards were consequential, ensuring that participants needed to trade-off exploration and exploitation. Participants had to find options that give them the most money while having to minimize sampling from low reward options. The N-armed bandits is well studied and it affords detailed analysis of information processing in terms of continuation and switching behavior. Theoretically, expectations of future reward should rise with adequate, but not excessive, exploration. Such "smart" exploration requires one to have a good representation of the task and its structure, which typically weighs already observed rewards by the degree to which an option has been explored. This would thus involve an internal model of the task contingency between average payoff and average payoff uncertainty (see also Worthy, Cooper, Byrne, Gorlick, & Maddox, 2014). Good performance in the task thus depends on learning mechanisms that use adequate future reward representations while performance anomalies will involve inadequate future reward representations.

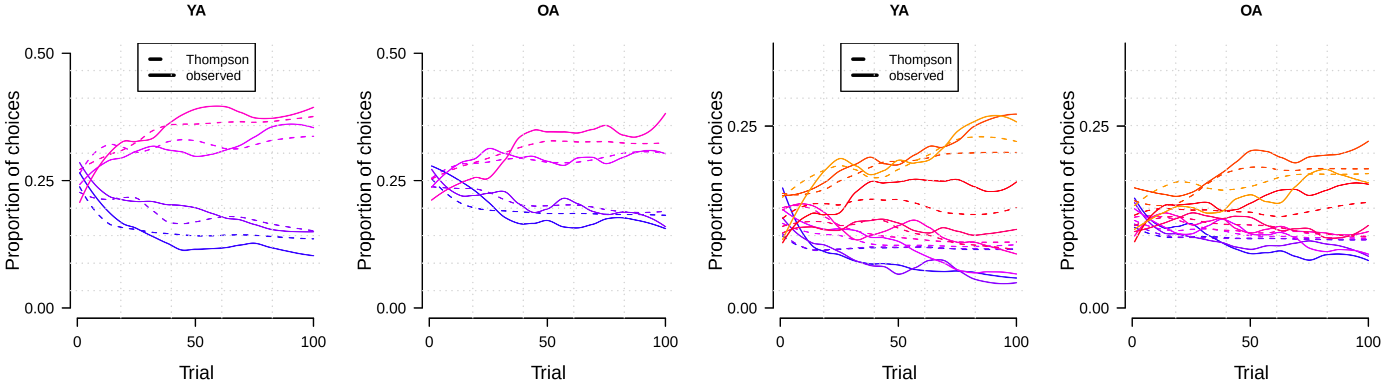
We hypothesized that OA achieve lower performance and arrive slower at the higher reward options, depending on the number of options. If OA focus more on reward in current states, their rewards in future states should suffer. Such short-term planning would more closely resemble experience-based learning rather than goal-directed learning. OA not arriving at the higher reward options at all would show a lack of an explore-exploit trade-off. We assumed that problems with a larger number of options are relatively more cognitively demanding because of a larger search space, which increases the amount of necessary information processing and representation. Next, we describe methods and results from six kinds of data analyses: choice proportions statistics, choice proportions over trials, regret over trials, comparisons to an optimality model, comparisons to a fitted optimality model, and one-step ahead predictions of a fitted optimality model. We end with a discussion.

2 Methods

Participants 32 older adults (OA, M_{age} = 70.5, 65-74, 38% female) and 29 younger adults (YA, M_{age} = 24.3, 19-30, 45% female) participated in this study. All participants were healthy, right-handed, native German speakers with normal or corrected to normal vision, and without a history of psychiatric or neurological disorders. There were no group differences in gender proportion, educational level, and socio-economic status. Compensation amounted to about 10 Euro per hour, plus on average 2 Euro performance-dependent bonus. Participants were recruited using advertisements.

Task The task of the participants was to maximize the sum of rewards in a total of 16 alternating four and eight-armed bandit problems. Rewards could be earned by selecting pictures of casino-style gambling machines presented on a computer screen using a keyboard or mouse. The gambling machines provided random rewards (1 or 0) with a hidden probability that was specific to each machine. The rewards were displayed on the respective bandit after each play. Participants had 100 trials for every problem to explore the hidden reward probabilities and to exploit those machines that give rewards most often. Remaining trials were displayed on the screen. Also, every bandit showed the number of plays so far and the probability of a reward based on the observed rewards so far. This information is sufficient to make an optimal choice at any point in time, reducing the role of working memory. Of course, participants still need to figure out how they want to trade off exploration and exploitation. 89% of YA and 70% of OA (p = .14, test of equal proportions) indicated in a post-task questionnaire usage of "the extra information regarding the options".

Procedure Participants were instructed 1) to maximize the sum of rewards, 2) how the task looked and worked, 3) that each trial is independently generated, and 4) that the best gambling machine in every individual problem had $p_{opt} = .6$ (to help comparability



(a) Four options

(b) Eight options

Figure 1: Observed choice proportions and one-step ahead predictions for fitted Thompson sampling. The color of the lines corresponds to the value of the hidden probabilities, where blue colors represent lower probabilities. Local polynomial regression was used as a moving window to smooth the trajectories (using a neighborhood of 40% of all points where neighboring points are also being weighted by their distance tricubically)

across problems). All participants had taken part in an unrelated fMRI study on risk-taking preference several weeks beforehand. Ethics approval was granted by the Institutional Review Board of the Max Planck Institute for Human Development.

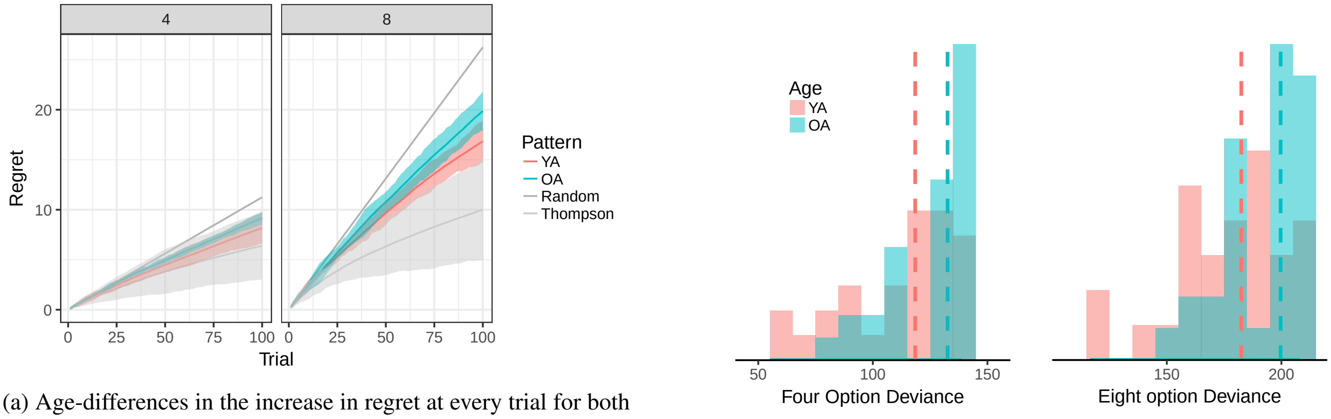
Design The experiment made use of a repeated within-subject condition (four vs eight options), and a between-subject condition (age group). We chose the other hidden probabilities in steps of .075 below .6. Reliably finding the better options thus required a significant part of exploration out of the 100 available trials. All participants saw the same randomly generated rewards for all 16 problems. This allowed comparison of the problem difficulty across participant groups as well as a reduction of an unnecessary source for variance in performance while keeping the probabilistic character of the task intact. Four different problem orders were generated and counterbalanced across participants. Two different orders started with four options and two different orders started with eight options. Between problems, performance was displayed on the screen and a keypress was required to continue with the next problem. Participants in both groups took about half an hour to finish the experiment. The minimum response time was set to 200 milliseconds.

3 Results

We first investigated age-related differences in task performance. Proportions of choices for each option revealed that YA sampled reliably more often from the best option. OA chose the highest-expected-value option about 5% less often than YA did in both four and eight option conditions (four option 95% HDI: .003 - .093; eight option 95% HDI: .018 - .096; Bayesian ANOVA with logit function and broad prior). We also tested a linear mixed effects model of the hidden probability of a chosen bandit with participant ID, problem ID, and bandit position ID as random effects, using Satterthwaite's approximations of p-values (*** indicating $p < .0001$). We found negative interaction effects for OA in eight options (B = -.021***) and for OA in eight options over trials (B = -.011***). Together, these indicated increasingly lower performance for OA in eight options over trials. We also found a positive interaction effect for both YA and OA in eight options over trials (B = .021***), as participants could improve relatively more over time for eight options. There still was a main negative effect of eight options (B = -.092***) and a main positive effect of trial number (B = -.009***). No significant difference or decrease over trials for OA remained, so the age effect is captured only by the higher-level interactions.

Second, we examined development of age-differences in choice proportions over trials, see Figures 1a and 1b. For every trial, the solid lines represent the average number of times that participants chose an option. The local instabilities in the trajectories may result from individual differences and variation across the several problems. On average, the third best option stops overlapping with the second best option after about 25 trials for YA. For OA, the same separation exists between the second and third option after twice as many trials, see the right panel of Figure 1a. In the eight option condition, YA separate between the better three options and the worse five options after about 50 trials. OA do this after about 75 trials. These 2-2 and 3-5 separations could reflect the participants' psychologically most salient explore-exploit representations. Together, the choice trajectories show that already from the beginning onwards, YA choose more often from the better options.

Third, we analyzed another measure of performance to compare performance across all of the options at once. We choose to measure regret (a common measure in machine learning) as it generalizes over the specific outcomes of the random number generation process. Regret can be computed as $R_T = \sum_{i=1}^{100} \left(p_{opt} - p_{B(i)}\right)$, where $p_{opt} = .6$ and $p_{B(i)}$ is the hidden probability of a reward for the chosen



(a) Age-differences in the increase in regret at every trial for both groups with standard deviations displayed around the means (standard errors were too narrow to visualize). Regret always starts at 0, increases quickly first, slower later on for both groups, but slowest for YA.

(b) Histogram and medians of median deviance per individual across games from fitted Thompson sampling

Figure 2: Task performance for the four and eight option conditions.

bandit. It follows that randomly behaving agents get a total regret of 11.25 points for four options and 26.25 points for eight options. Overall, the age effect on regret was large ($p < .01$, Cohen's d .707) for eight options (M_{OA} = 19.87, *SE* = .79, M_{YA} = 16.84, *SE* .75) and medium ($p < .05$, Cohen's d .550) for four options (M_{OA} = 9.16, *SE* = .32, M_{YA} = 8.17, *SE* .33). We also investigated how regret differences emerged using the shapes of the exploration-exploitation trade-offs over trials within the choice profiles. We observed a slowing increase in regret over time in general but increasing age-related differences for both conditions, see Figure 2a. It seems that OA need longer exploration than YA. For both groups, average regret was reliably better compared to a random agent (and reliably worse compared to an optimal agent from less than 20 trials, see next paragraph).

Fourth, we wanted to know how participant performance differed from optimality. We used Thompson sampling as an optimality model (Thompson, 1933), but we observed that differences in regret for similar algorithms are small in the context of the present task. Thompson sampling uses an inverse cumulative distribution function (also known as percentage point function or quantile function) that is used to choose the bandit with the highest certainty that the hidden probability of a bandit is smaller or equal than some randomly generated value. This way, the algorithm minimizes uncertainty that there exists a better option by making sure that the probability of choosing a certain bandit is proportional to the probability of it being the best bandit. By taking uncertainty into account, the algorithm affords a way of more rapidly adapting its decision if not only the mean of a certain bandit gets overtaken by another mean, but the whole posterior probability distribution. Conceptually, the algorithm keeps track of beliefs about the hidden probabilities of the bandits and then updates these beliefs each time after seeing an outcome. The algorithm is initialized by setting a uniform prior for all options. The algorithm then plays option x proportional to the probability of it being the best. Finally, it updates its priors using the newly collected information. Thompson sampling achieved an expected regret of 6.5 points for four options and 11.0 points for eight options, which is considerably better than YA performed on average (170% larger than the gap between YA and OA for four options and 193% for eight options). Interestingly, 5 out of 32 OA (16%) and 9 out of 29 YA (31%) achieved a median regret score within 10% of Thompson sampling for four options, while this was 1 (3%) and 4 (14%) for eight options. Some individuals were thus able to achieve regret scores similar to Thompson sampling.

Fifth, we wanted to know how well a fitted optimality model predicted participant's decisions. Thompson sampling was fitted to individual games by scaling predicted choices using a softmax function with a fitted inverse temperature parameter θ, ranging from 0.003 to 30 (higher θ values produced more randomness). OA deviated more from fitted Thompson sampling than YA did ($p < .05$, Wilcoxon tests, Cohen's d = .57 for four options and Cohen's d = .53 for eight options), see Figure 2b. θ was also significantly lower for YA than for OA ($p < .05$, Wilcoxon test, Cohen's d = .96 for four and 2.68 for eight options), indicating more randomness and worse matches to predictions of Thompson sampling in OA than in YA. OA and YA both significantly decreased their median θ for eight options compared to the four option condition ($p < .05$, Wilcoxon tests). θ for OA significantly varied more in both conditions than for YA ($p < .01$ for both conditions, Wilcoxon tests) and average variation across games was significantly lower in four options for YA, but for OA this was similar in both conditions ($p < .01$ vs. $p = .4$, Wilcoxon tests). In all, OA adults were more random and less homogeneous, possibly indicating more strategy changes.

Finally, we compared the shapes of the mean observed exploration-exploitation trade-off trajectories to shapes from one-step-ahead predictions across all trials. These predictions are plotted in Figures 1a and 1b using dashed lines. We used the median θ of every participant for both conditions as data scaling parameter. The predictions from this fitted Thompson sampling model resulted in accurately ordered trajectories for both groups and for both conditions: The orderings of solid and dashed lines were identical for all four graphs for most trials, except in the first few trials. The latter may indicate more rapid exploration in Thompson sampling and that both YA and OA explore less rapidly, with OA taking the longest.

4 Discussion and Conclusions

We aimed to identify changes in the ways OA and YA make goal-directed choices depending on the complexity of the choice environment. We found a large age-related effect on performance in a typical eight-armed bandit task and a smaller effect in a four-armed bandit task. YA also deviated less from optimality than OA did. Choice trajectories showed that age effects were already observable in the early exploration stage, suggesting that OA explore longer or less efficiently. Theoretically, the early stages require fast exploration using not only average rewards but also their associated uncertainty. This was illustrated here using Thompson sampling, which is a kind of randomized probability matching algorithm. Participants diversified their choices similar to Thompson sampling, in line with previous work (Konstantinidis, Ashby, & Gonzalez, 2015; Speekenbrink & Konstantinidis, 2015). Furthermore, OA had higher and more variable inverse temperature parameter estimates across choice environments, indicating more randomness in OA. OA thus rely on less effective learning strategies that consider important information less effectively, in particular in the more complex environments.

Why would OA fail to represent important information like uncertainty or a specific task model? The role of working memory influences should be minimal as this is not strictly necessary to perform well in the task. General "slowing", gender effects, and more cautious risk taking, all of which could favor exploitation of short-term rewards, also mark cognitive aging. We did indeed observe differences in reaction times as well as in standard neuropsychological test results (working memory, fluid intelligence, and risk-taking). However, as performance is mainly determined by a cognitively costly explore-exploit tradeoff and adequate future reward representations, our findings specifically point towards underreliance on goal-directed learning.

A logical next step is to assess if fits of simple learning strategies can indeed better accommodate OA. Specifically, the exploration phase seems to happen sub-optimally in OA and in a more varied way. Favoring short-term rewards could be a sign of a learning mechanism that sub-optimally represents future rewards. More varied or reduced processing of important information such as uncertainty would be able to account successfully for the observed age-related changes. Furthermore, if the task indeed probes OA to rely less on goal-directed learning, we may also expect differences in connectivity to prefrontal regions (pending analyses). In all, OA may be using less effective learning strategies the more demanding the choice environment becomes. Identifying such task-dependent differences is typically neglected in neuro-computational models of decision-making. In the context of cognitive aging, this may be useful for empowering aging decision makers to navigate cognitively demanding choice environments.

5 References

Chowdhury, R., Guitart-Masip, M., Lambert, C., Dayan, P., Huys, Q., Dzel, E., & Dolan, R. J. (2013). Dopamine restores reward prediction errors in old age. *Nature Neuroscience*, *16*, 648–653.

Daw, N. D., & O'Doherty, J. P. (2014). Chapter 21 - Multiple Systems for Value Learning. In P. W. G. Fehr (Ed.), *Neuroeconomics (Second Edition)* (pp. 393–410). San Diego: Academic Press.

Eppinger, B., & Bruckner, R. (2015). Chapter 4 - Towards a Mechanistic Understanding of Age-Related Changes in Learning and Decision Making: A Neuro-Computational Approach. In T. M. Hess & J. S. E. Lckenhoff (Eds.), *Aging and Decision Making* (pp. 61–77). San Diego: Academic Press.

Eppinger, B., Walter, M., Heekeren, H. R., & Li, S.-C. (2013). Of goals and habits: Age-related and individual differences in goal-directed decision-making. *Frontiers in Neuroscience*, *7*. doi:10.3389/fnins.2013.00253

Frey, R., Mata, R., & Hertwig, R. (2015). The role of cognitive abilities in decisions from experience: Age differences emerge as a function of choice set size. *Cognition*.

Konstantinidis, E., Ashby, N. J., & Gonzalez, C. (2015). Exploring Complexity in Decisions from Experience: Same Minds, Same Strategy. In *37th annual meeting of the Cognitive Science Society (CogSci 2015)* (pp. 23–25).

Mata, R., Pachur, T., Von Helversen, B., Hertwig, R., Rieskamp, J., & Schooler, L. (2012). Ecological rationality: A framework for understanding and aiding the aging decision maker. *Decision Neuroscience*, *6*, 19.

Mata, R., Schooler, L. J., & Rieskamp, J. (2007). The aging decision maker: Cognitive aging and the adaptive selection of decision strategies. *Psychology and Aging*, *22*, 796–810.

Samanez-Larkin, G. R., & Knutson, B. (2015). Decision making in the ageing brain: Changes in affective and motivational circuits. *Nature Reviews Neuroscience*, *advance online publication*. doi:10.1038/nrn3917

Speekenbrink, M., & Konstantinidis, E. (2015). Uncertainty and Exploration in a Restless Bandit Problem. *Topics in Cognitive Science*, *7*, 351–367.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 285–294.

Worthy, D. A., Cooper, J. A., Byrne, K. A., Gorlick, M. A., & Maddox, W. T. (2014). State-based versus reward-based motivation in younger and older adults. *Cognitive, Affective, & Behavioral Neuroscience*, *14*, 1208–1220.

Efficient Parallel Methods for Deep Reinforcement Learning

Alfredo V. Clemente
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
`alfredvc@stud.ntnu.no`

Humberto N. Castejón
Telenor Research
Trondheim, Norway
`humberto.castejon@telenor.com`

Arjun Chandra
Telenor Research
Trondheim, Norway
`arjun.chandra@telenor.com`

Abstract

We propose a novel framework for efficient parallelization of deep reinforcement learning algorithms, enabling these algorithms to learn from multiple actors on a single machine. The framework is algorithm agnostic and can be applied to on-policy, off-policy, value based and policy gradient based algorithms. Given its inherent parallelism, the framework can be efficiently implemented on a GPU, allowing the usage of powerful models while significantly reducing training time. We demonstrate the effectiveness of our framework by implementing an advantage actor-critic algorithm on a GPU, using on-policy experiences and employing synchronous updates. Our algorithm achieves state-of-the-art performance on the Atari domain after only a few hours of training. Our framework thus opens the door for much faster experimentation on demanding problem domains.

Keywords: Deep Neural Networks, Reinforcement Learning, Parallel Computing

1 Introduction and Related Work

Incorporating deep learning models within reinforcement learning (RL) presents some challenges. Standard stochastic gradient descent algorithms for training deep learning models assume all training examples to be independent and identically distributed (i.i.d.). This constraint is often violated by RL agents, given the high correlation between encountered states. Additionally, when learning on-policy, the policy affects the distribution of encountered states, which in turn affects the policy, creating a feedback loop that may lead to divergence [5]. Two main attempts have been made to solve these issues. One is to store experiences in a large replay memory and employ off-policy RL methods [5]. Sampling the replay memory can break the state correlations, thus reducing the effect of the feedback loop. Another is to execute multiple asynchronous agents in parallel, each interacting with an instance of the environment independently of each other [6]. Both of these approaches suffer from different drawbacks; experience replay can only be employed with off-policy methods, while asynchronous agents can perform inconsistent parameter updates due to stale gradients[1] and simultaneous parameter updates from different threads.

Parallel and distributed compute architectures have motivated innovative modifications to existing RL algorithms to efficiently make use of parallel execution. In the General Reinforcement Learning Architecture (Gorila) [8], the DQN [7] algorithm is distributed across multiple machines. Multiple *learners* learn off-policy using experiences collected into a common replay memory by multiple *actors*. Gorila is shown to outperform standard DQN in a variety of Atari games, while only training for 4 days. The distribution of the learning process is further explored in [6], where multiple actor-learners are executed asynchronously on a single machine. Each actor-learner holds its own copy of the policy/value function and interacts with its own instance of the environment. This allows for both off-policy, as well as on-policy learning. The actor-learners compute gradients in parallel and update shared parameters asynchronously in a Hogwild! [10] fashion. The authors suggest that multiple agents collecting independent experiences from their own environment instances reduces correlation between samples, thereby improving learning. The asynchronous advantage actor-critic (A3C) algorithm [6] was able to surpass the state of the art on the Atari domain at the time of publication, while training for 4 days on a single machine with 16 CPU cores. GA3C [2] is a GPU implementation of A3C. It batches action selection and learning using queues. Actors sample from a shared policy by submitting a task to a *predictor*, which executes the policy and returns an action once it has accumulated enough tasks. Similarly, learning is performed by submitting experiences to a *trainer*, which computes gradients and applies updates once enough experiences have been gathered. If the training queue is not empty when the model is updated, the learning will no longer be on-policy, since the remaining experiences were generated by an old policy. This leads to instabilities during training, which the authors address with a slight modification to the weight updates.

We propose a novel framework for efficient parallelization of deep reinforcement learning algorithms, which keeps the strengths of the aforementioned approaches, while alleviating their weaknesses. Algorithms based on this framework can learn from hundreds of actors in parallel, similar to Gorila, while running on a single machine like A3C and GA3C. Having multiple actors help decorrelate encountered states and attenuate feedback loops, while allowing us to leverage the parallel architecture of modern CPUs and GPUs. Unlike A3C and Gorila, there is only one copy of the parameters, hence parameter updates are performed synchronously, thus avoiding the possible drawbacks related to asynchronous updates. Our framework has many similarities to GA3C. However, the absence of queues allows for a much simpler and computationally efficient solution, while allowing for true on-policy learning and faster convergence. We demonstrate our framework with a Parallel Advantage Actor-Critic algorithm, that achieves state of the art performance in the Atari 2600 domain after only a few hours of training, opening the door for much faster experimentation.

2 Background

2.1 Batching with Stochastic Gradient Descent

Current reinforcement learning algorithms make heavy use of deep neural networks, both to extract high level features from the observations it receives, and as function approximators to represent its policy or value functions.
Consider the set of input-target pairs $S = \{(x_0, y_0), (x_1, y_1), ...(x_n, y_n)\}$ generated by some function $f^*(x)$. The goal of supervised learning with neural networks is to learn a parametrized function $f(x;\theta)$ that best approximates function $f*$. The performance of f is evaluated with the empirical loss

$$L(\theta) = \frac{1}{|S|}\sum_{s\in S} l(f(x_s;\theta), y_s) \tag{1}$$

where $l(f(x_s;\theta), y_s)$ is referred to as a loss function, and gives a quantitative measure of how good f is at modelling f^*. The model parameters θ are learned with stochastic gradient descent (SGD) by iteratively applying the update $\theta_{i+1} \leftarrow \theta_i - \alpha\nabla_{\theta_i} L(\theta_i)$ for a learning rate α. In SGD, $L(\theta)$ is usually approximated with $\bar{L}(\theta) = \frac{1}{|S'|}\sum_{s'\in S'} l(f(x_{s'};\theta), y_{s'})$,

---

[1] Gradients may be computed w.r.t. stale parameters while updates applied to a new parameter set.

where $S' \subset S$ is a mini-batch sampled from S. The choice of α and $n_{S'} = |S'|$ presents a trade-off between computational efficiency and sample efficiency. Increasing $n_{S'}$ by a factor of k increases the time needed to calculate $\nabla_\theta \bar{L}$ by a factor of k', for $k' \leq k$, and reduces its variance proportionally to $\frac{1}{k}$ [4]. In order to mitigate the increased time per parameter update we can increase the learning rate to α' for some $\alpha' \geq \alpha$. However there are some limits on the size of the learning rate, so that in general $\alpha' \leq k\alpha$ [4]. The hyper-parameters α' and k are chosen to simultaneously maximize $\frac{k}{k'}$ and minimize L.

3 Parallel Framework for Deep Reinforcement Learning

We propose a general framework for deep reinforcement learning, where multiple actors can be trained synchronously on a single machine. A set of n_e environment instances are maintained, where actions for all environment instances are generated from the policy. The architecture can be seen in Figure 1. The policy function can be represented implicitly, as in value based methods, or explicitly as in policy gradient methods. As suggested in Mnih et al. [6], by having multiple environments instances in parallel it is likely that they will be exploring different locations of the state space at any given time, which reduces the correlation of encountered states and helps stabilize training. This approach can be motivated as an *on-line* experience memory, where experiences are sampled from the distribution currently being observed from the environment, instead of sampling uniformly from previous experience. At each timestep the master generates actions for all environment instances by sampling the current policy. Note, that the policy may be sampled differently for each environment. A set of n_w workers then apply all the actions to the their respective environments in parallel, and store the observed experiences. The master then updates the policy with the observed experiences. This allows the evaluation and training of the policy to be batched, which can be efficiently parallelized, leading to significant speed improvements on modern compute architectures.

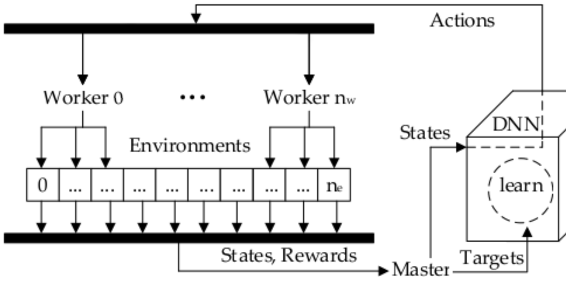


Figure 1: Architecture of the Parallel Framework for Deep RL

4 Parallel Advantage Actor Critic

We used the proposed framework to implement a version of the n-step advantage actor-critic algorithm proposed by Mnih et al. [6]. This algorithm maintains a policy $\pi(a_t|s_t;\theta)$ and an estimate $V(s_t;\theta_v)$ of the value function, both approximated by deep neural networks. The parameters θ of the policy network (the actor) are optimized via gradient ascent following $\nabla_\theta \log \pi(a_t|s_t;\theta) A(s_t, a_t; \theta, \theta_v) + \beta \nabla_\theta H(\pi(s_{e,t};\theta))$ [11], where $A(s_t, a_t; \theta, \theta_v) = Q^{(n)}(s_t, a_t; \theta, \theta_v) - V(s_t; \theta_v)$ is an estimate of the advantage function, $Q^{(n)}(s_t, a_t; \theta, \theta_v) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n}; \theta_v)$, with $0 < n \leq t_{max}$, is the *n*-step return estimation and $H(\pi(s_{e,t};\theta))$ is the entropy of the policy π, which, as suggested by Mnih et al. [6], is added to improve exploration by discouraging premature convergence to suboptimal deterministic policies. The parameters θ_v of value network (the critic) are in turn updated via gradient descent in the direction of $\nabla_{\theta_v} \left[\left(Q^{(n)}(s_t, a_t; \theta, \theta_v) - V(s_t; \theta_v) \right)^2 \right]$.

In the context of our framework, the above gradients are calculated using mini batches of experiences. For each of the n_e environments, t_{max} experiences are generated, resulting in batches of size $n_e \cdot t_{max}$. The gradients ∇_θ^π for the policy network and the gradients ∇_θ^V for the value function thus take the following form:

$$\nabla_\theta^\pi \approx \frac{1}{n_e \cdot t_{\max}} \sum_{e=1}^{n_e} \sum_{t=1}^{t_{\max}} \left(Q^{(t_{\max}-t+1)}(s_{e,t}, a_{e,t}; \theta, \theta_v) - V(s_{e,t}; \theta_v) \right) \nabla_\theta \log \pi(a_{e,t}|s_{e,t};\theta) + \beta \nabla_\theta H(\pi(s_{e,t};\theta)) \tag{2}$$

$$\nabla_{\theta_v}^V \approx \nabla_{\theta_v} \frac{1}{n_e \cdot t_{max}} \sum_{e=1}^{n_e} \sum_{t=1}^{t_{\max}} \left(Q^{(t_{\max}-t+1)}(s_{e,t}, a_{e,t}; \theta, \theta_v) - V(s_{e,t}; \theta_v) \right)^2 \tag{3}$$

Pseudocode for our parallel advantage actor-critic algorithm (PAAC) is given in Algorithm 1. As shown in the next section, PAAC achieves state of the art performance on the Atari 2600 domain in half of the time required by GA3C and in only one eigth of the time required by A3C. Note that, although we implement an actor-critic algorithm, this framework can be used to implement any other reinforcement learning algorithm.

5 Experiments

We evaluated the performance of PAAC in 12 games from Atari 2600 using the Atari Learning Environment [3]. The agent was developed in Python using TensorFlow [1] and all performance experiments were run on a computer with a 4 core Intel i7-4790K processor and an Nvidia GTX 980 Ti GPU.

5.1 Experimental setup

To compare results with other algorithms for the Atari domain we follow the same pre-processing and training procedures as Mnih et al. [6]. Each action is repeated 4 times, and the per-pixel maximum value from the two latest frames is kept. The frame is then scaled down from 210×160 pixels and 3 color channels to 84×84 pixels and a single color channel for pixel intensity. Whenever an environment is restarted, the state is reset to the starting state and between 1 and 30 no-op actions are performed before giving control to the agent. The environment is restarted whenever the final state of the environment is reached.

As in [6], a single convolutional network with two separate output layers was used to jointly model the policy and the value functions. For the policy function, the output is a softmax with one node per action, while for the value function a single linear output node is used. Moreover, to compare the efficiency of PAAC for different model sizes, we implemented two variants of the policy and value convolutional network. The first variant, referred to as $\text{arch}_{\text{nips}}$, is the same architecture used by A3C FF [6], which is a modified version of the architecture used in Mnih et al. [5], adapted to an actor-critic algorithm. The second variant, $\text{arch}_{\text{nature}}$, is an adaptation of the architecture presented in Mnih et al. [7]. The networks were trained with RMSProp. The hyperparameters used to generate the results in Table 1 were $n_w = 8$, $n_e = 32$, $t_{\max} = 5$, $N_{\max} = 1.15 \times 10^8$, $\gamma = 0.99$, $\alpha = 0.0224$, $\beta = 0.01$, and a discount factor of 0.99 and epsilon of 0.1 for RMSProp. Additionally gradient clipping [9] with a threshold of 40 was used.

Algorithm 1 Parallel advantage actor-critic

```
Initialize timestep counter $N = 0$ and network weights $\theta, \theta_v$
Instantiate set $\boldsymbol{e}$ of $n_e$ environments
repeat
    for $t = 1$ to $t_{max}$ do
        Sample $\boldsymbol{a}_t$ from $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t; \theta)$
        Calculate $\boldsymbol{v}_t$ from $V(\boldsymbol{s}_t; \theta_v)$
        parallel for $i = 1$ to $n_e$ do
            Perform action $a_{t,i}$ in environment $e_i$
            Observe new state $s_{t+1,i}$ and reward $r_{t+1,i}$
            Store $f_{t+1,i} = \begin{cases} 0 & \text{for terminal } \boldsymbol{s}_t \\ 1 & \text{for non-terminal } \boldsymbol{s}_t \end{cases}$
        end parallel for
    end for
    $\boldsymbol{R}_{t_{\max}+1} = V(s_{t_{\max}+1}; \theta)$
    for $t = t_{\max}$ down to 1 do
        $\boldsymbol{R}_t = \boldsymbol{r}_t + \gamma \boldsymbol{R}_{t+1} \odot \boldsymbol{f_{t+1}}$
    end for
    $d\theta = \frac{1}{n_e \cdot t_{max}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{max}} (R_{t,i} - v_{t,i}) \nabla_\theta \log \pi(a_{t,i}|s_{t,i}; \theta) + \beta \nabla_\theta H(\pi(s_{e,t}; \theta))$
    $d\theta_v = \frac{1}{n_e \cdot t_{max}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{max}} \nabla_{\theta_v} (R_{t,i} - V(s_{t,i}; \theta_v))^2$
    Update $\theta$ using $d\theta$ and $\theta_v$ using $d\theta_v$.
    $N \leftarrow N + n_e \cdot t_{\max}$
until $N \geq N_{max}$
```

5.2 Results

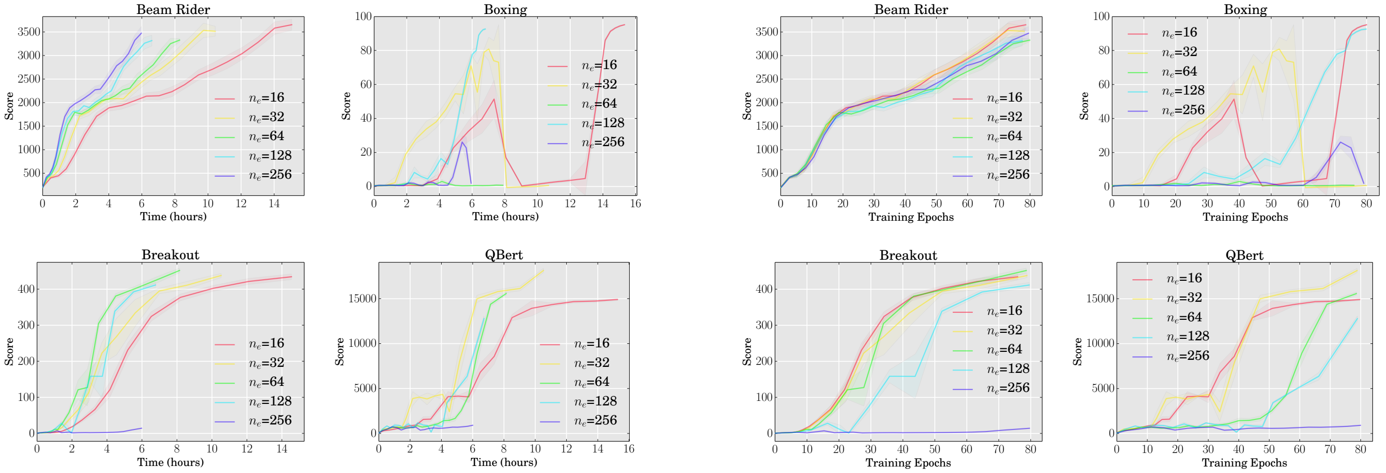
The performance of PAAC with $\text{arch}_{\text{nips}}$ and $\text{arch}_{\text{nature}}$ was evaluated on twelve different Atari 2600 games, where agents were trained for 115 million skipped frames (460 million actual frames). The results and their comparison to Gorila [8], A3C [6] and GA3C [2] are presented in Table 1. After a few hours of training on a single computer, PAAC is able to outperform Gorila in 8 games, and A3C FF in 8 games. Of the 9 games used to test GA3C, PAAC matches its performance in 2 of them and surpasses it in the remaining 7. To better understand the effect of the number of actors (and batch size) on the score, tests were run with $n_e \in \{16, 32, 64, 128, 256\}$. The learning rate was not tuned for each batch size, and was chosen to be $0.0007 \cdot n_e$ for all runs across all games. Increasing n_e decreases the frequency of parameter updates, given that parameter updates are performed every $n_e \cdot t_{\max}$ timesteps. As the theory suggests, the decreased frequency in parameter updates can be offset by increasing the learning rate. As can be seen in Figure 2b most choices of n_e result in similar scores at a given timestep,

Game	Gorila	A3C FF	GA3C	PAAC $\text{arch}_{\text{nips}}$	PAAC $\text{arch}_{\text{nature}}$
Amidar	1189.70	263.9	218	701.8	1348.3
Centipede	8432.30	3755.8	7386	5747.32	7368.1
Beam Rider	3302.9	22707.9	N/A	4062.0	6844.0
Boxing	94.9	59.8	92	99.6	99.8
Breakout	402.2	681.9	N/A	470.1	565.3
Ms. Pacman	3233.50	653.7	1978	2194.7	1976.0
Name This Game	6182.16	10476.1	5643	9743.7	14068.0
Pong	18.3	5.6	18	20.6	20.9
Qbert	10815.6	15148.8	14966.0	16561.7	17249.2
Seaquest	13169.06	2355.4	1706	1754.0	1755.3
Space Invaders	1883.4	15730.5	N/A	1077.3	1427.8
Up n Down	12561.58	74705.7	8623	88105.3	100523.3
Training	4d CPU cluster	4d CPU	1d GPU	12h GPU	15h GPU

Table 1: Scores are measured from the best performing actor out of three, and averaged over 30 runs with upto 30 no-op actions start condition. Results for A3C FF use human start condition are therefore not directly comparable. Results for the Random scores taken from [3], Gorila scores taken from Nair et al. [8], A3C FF scores taken from Mnih et al. [6] and GA3C scores take from Babaeizadeh et al. [2]. Unavailable results are shown as N/A.

however Figure 2a shows that higher values of n_e reach those timesteps significantly faster. The choice of $n_e = 256$ results in divergence in three out of the four games, which shows that the learning rate can be increased proportional to the batch size, until a certain limit is reached.

A limiting factor in the speed of training is the time spent in agent-environment interaction. When using $\text{arch}_{\text{nips}}$ for $n_e = 32$ approximately 50% of the time is spent interacting with the environment, while only 37% is used for learning and action selection, as is shown in Figure 3. This has strong implications for the models and environments that can be used. Using a model-environment combination that doubles the time needed for learning and action calculation would lead to a mere 37% increase in training time. This can be seen in Figure 3 where using $\text{arch}_{\text{nature}}$ on the GPU leads to a drop in timesteps per second 22% for $n_e = 32$ when compared to $\text{arch}_{\text{nips}}$. When running on the CPU however this leads to a 41% drop in timesteps per second.



(a) Score achieved per wall time.

(b) Score achieved per epoch, where one training epoch is equivalent to 1 million timesteps (4 million frames)

Figure 2: Scores achieved by PAAC on four Atari 2600 games for different values of n_e.

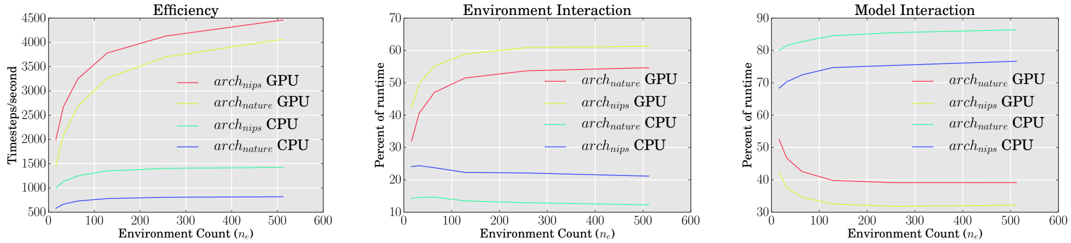


Figure 3: Time usage in the game of Pong for different n_e.

6 Conclusion

In this work, we have introduced a parallel framework for deep reinforcement learning that can be efficiently parallelized on a GPU. The framework is flexible, and can be used for on-policy and off-policy, as well as value based and policy gradient based algorithms. The presented implementation of the framework is able to reduce training time for the Atari 2600 domain to a few hours, while maintaining state-of-the-art performance. Improvements in training time, will allow the application of these algorithms to more demanding environments, and the use of more powerful models.

References

[1] Abadi, Martın, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow. org*, 1, 2015.

[2] Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., and Kautz, J. GA3C: GPU-based A3C for Deep Reinforcement Learning. *ArXiv e-prints*, November 2016.

[3] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 06 2013.

[4] Bottou, L., Curtis, F. E., and Nocedal, J. Optimization Methods for Large-Scale Machine Learning. *ArXiv e-prints*, June 2016.

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *ArXiv e-prints*, December 2013.

[6] Mnih, V., Puigdomènech Badia, A., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. *ArXiv e-prints*, February 2016.

[7] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[8] Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., Legg, S., Mnih, V., Kavukcuoglu, K., and Silver, D. Massively Parallel Methods for Deep Reinforcement Learning. *ArXiv e-prints*, July 2015.

[9] Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training Recurrent Neural Networks. *ArXiv e-prints*, November 2012.

[10] Recht, Benjamin, Re, Christopher, Wright, Stephen, and Niu, Feng. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pp. 693–701, 2011.

[11] Sutton, Richard S, McAllester, David A, Singh, Satinder P, Mansour, Yishay, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pp. 1057–1063, 1999.

Combining Neural Networks and Tree Search for Task and Motion Planning in Challenging Environments

Chris Paxton
Zoox, Inc.
Menlo Park, CA 94025
chris.paxton@zoox.com

Vasumathi Raman
Zoox, Inc.
Menlo Park, CA 94025
vasu@zoox.com

Gregory D. Hager
Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218
hager@cs.jhu.edu

Marin Kobilarov
Zoox, Inc.
Menlo Park, CA 94025
marin@zoox.com

Abstract

Robots in the real world have to deal with complex scenes involving multiple actors and complex, changing environments. In particular, self-driving cars are faced with a uniquely challenging task and motion planning problem that incorporates logical constraints with multiple interacting actors in a scene that includes other cars, pedestrians, and bicyclists.

A major challenge in this setting, both for neural network approaches and classical planning, is the need to explore future worlds of a complex and interactive environment. To this end, we integrate Monte Carlo Tree Search with hierarchical neural net control policies trained on expressive Linear Temporal Logic (LTL) specifications. We propose a methodology that incorporates deep neural networks to learn low-level control policies as well as high-level "option" policies. We thus investigate the ability of neural networks to learn both LTL constraints and continuous control policies in order to generate task plans.

We demonstrate our approach in a simulated autonomous driving setting, where a vehicle must drive down a road shared with multiple other vehicles, avoid collisions, and navigate an intersection, all while obeying given rules of the road.

This work has been submitted to a general robotics conference; it is our intention to communicate it more broadly to the reinforcement learning and decision making community.

Keywords: task and motion planning, linear temporal logic, formal methods, deep reinforcement learning

1 Introduction

A robot operating in the physical world must reason in a hybrid space: both its continuous motion in the physical world and the discrete goals it must accomplish are pertinent to correctly completing a complex task. One case where this is particularly relevant is in the domain of autonomous driving. Self-driving cars have to deal with a highly complex and dynamic environment: they share a road with other moving vehicles, as well as with pedestrians and bicyclists. Road conditions are also unpredictable, meaning that such methods must be capable of dealing with uncertainty.

The field of Task and Motion Planning (TAMP) seeks to integrate the solving of the continuous and discrete problems. Current methods succeed at solving many sequential path planning and spatial reasoning problems [10], but the combined discrete and continuous state space tends to explode in size for complex problems. The addition of temporal constraints makes the search problem even more difficult, though there has been recent progress in this direction [8]. On the other hand, recent work in Deep Reinforcement Learning (DRL) has shown promise in challenging domains including autonomous driving[2, 11], and has been combined with Monte Carlo Tree Search (MCTS) for game playing [9], where it was able to achieve master-level performance. However, the question remains open whether these approaches can be integrated to produce reliable robot behavior.

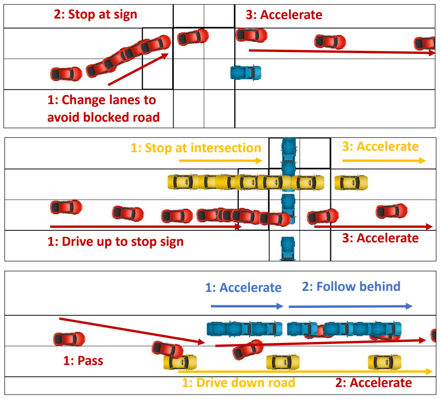


Figure 1: Solutions to simulated self driving car problems containing an intersection and multiple vehicles.

We show that the best of both worlds can be achieved by using neural networks to learn both low-level control policies and high-level action selection priors, and then using these multi-level policies as part of a heuristic search algorithm to achieve a complex task. We formulate task and motion planning as a variant of Monte Carlo Tree Search over high-level options, each of which is represented by a learned control policy, trained on a set of Linear Temporal Logic (LTL) formulae. LTL is an expressive language that has been used to concisely and precisely specify a wide range of system behaviors for robots. We refer the interested reader to [3] for further details. This approach allows us to efficiently explore the relevant parts of the search space to find high quality solutions when other methods would fail to do so. Fig. 1 shows some scenarios to which our algorithm was applied.

To summarize, our contributions are: (1) a planning algorithm combining learned low-level control policies with learned high level “option policies” over these low-level policies for TAMP in dynamic environments; (2) a framework for incorporating complex task requirements expressed in temporal logic; and (3) evaluation of the proposed approach in a simulated autonomous driving domain. Note that while our approach performs very well in simulation, it still has several limitations, which we discuss in Section 5. For more details, see the full version of the paper [7].

2 Approach

We consider systems that evolve according to continuous dynamics f_c and discrete dynamics f_d:

$$x' = f_c(x, w, u, o),\ w' = f_d(x, w, u, o)$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^n$ is the continuous state, $u \in \mathcal{U} \subseteq \mathbb{R}^m$ is the continuous control input, $w \in \mathcal{W}$ is the discrete (logical) world state, and $o \in \mathcal{O}$ is a discrete (logical) option from a finite set $\mathcal{O}$. Our atomic propositions $p \in AP$ are defined as functions over the discrete world state, i.e. $p : \mathcal{W} \to \{\texttt{True}, \texttt{False}\}$. In the MDP framework, $S = \mathcal{X} \times \mathcal{W}, A = \mathcal{U} \times \mathcal{O}, \delta(xw, uo) = x'w'$ such that $x' = f_c(x, w, u, o),\ w' = f_d(x, w, u, o)$. The labeling function over states is $\mathcal{L}(xw) = \{p \in AP$ such that $p(w) = \texttt{True}\}$.

We decompose the system into many actors. Each independent entity is an actor, and in particular the agent under control is itself an actor. A world state consists of an environment $e \in \mathcal{E}$ and some number of actors N. Actors represent other entities in the world that will update over time according to some policy specific to them.

Finally, we assume we are given a feature function $\phi : S \to \mathcal{F}$, which computes a low-dimensional representation of the world state containing all information needed to compute a policy. We then decompose the problem into finding two sets of policies: a policy $\pi_{\mathcal{O}} : \mathcal{F} \to \mathcal{O}$ over high-level actions and a policy $\pi_{\mathcal{U}} : \mathcal{O} \times \mathcal{F} \to \mathcal{U}$ over low-level controls, such that their composition solves the MDP. We compute a control policy for each high-level option o that maps from arbitrary feature values to controls and a second policy that over options, which tells us what we expect to be the best control policy to execute:

$$\pi^*_{\mathcal{U}}(\phi(xw), o) = \arg\max_u (V^*(\delta(xw, uo)))$$
$$\pi^*_{\mathcal{O}}(\phi(xw)) = \arg\max_o (V^*(\delta(xw, \pi^*_{\mathcal{U}}(\phi(xw), o)o)))$$

The hierarchical structure allows us to learn a reliable approximately-optimal policy with far less data than would otherwise be possible. Fig. 2 provides a schematic overview of our proposed approach.

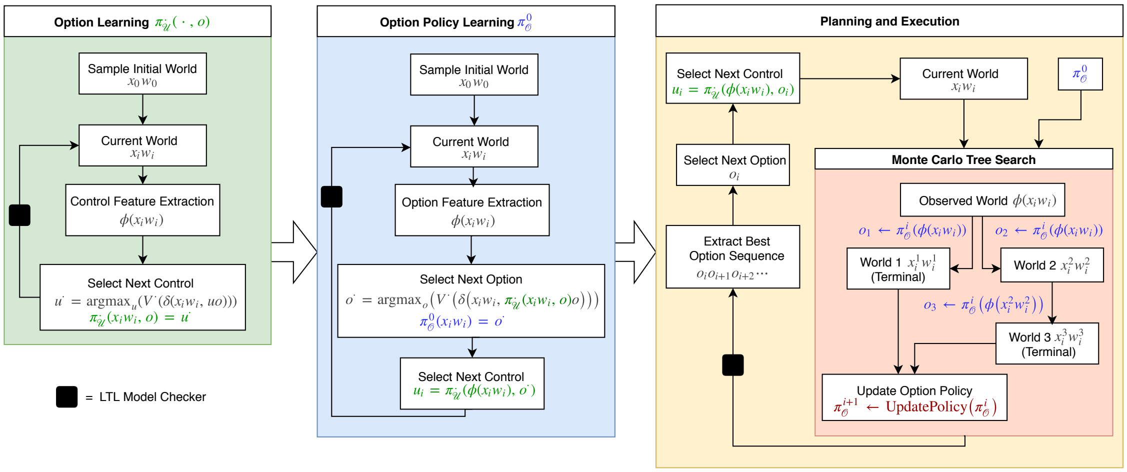

Option Learning $\pi^{\cdot}_{\mathcal{U}}(\cdot, o)$
Sample Initial World $x_0 w_0$
Current World $x_i w_i$
Control Feature Extraction $\phi(x_i w_i)$
Select Next Control $u^{\cdot} = \text{argmax}_u(V^{\cdot}(\delta(x_i w_i, uo)))$ $\pi^{\cdot}_{\mathcal{U}}(x_i w_i, o) = u^{\cdot}$
= LTL Model Checker
Option Policy Learning $\pi^0_{\mathcal{O}}$
Sample Initial World $x_0 w_0$
Current World $x_i w_i$
Option Feature Extraction $\phi(x_i w_i)$
Select Next Option $o^{\cdot} = \text{argmax}_o(V^{\cdot}(\delta(x_i w_i, \pi^{\cdot}_{\mathcal{U}}(x_i w_i, o)o)))$ $\pi^0_{\mathcal{O}}(x_i w_i) = o^{\cdot}$
Select Next Control $u_i = \pi^{\cdot}_{\mathcal{U}}(\phi(x_i w_i), o^{\cdot})$
Planning and Execution
Select Next Control $u_i = \pi^{\cdot}_{\mathcal{U}}(\phi(x_i w_i), o_i)$
Current World $x_i w_i$
$\pi^0_{\mathcal{O}}$
Select Next Option o_i
Extract Best Option Sequence $o_i o_{i+1} o_{i+2} \cdots$
Monte Carlo Tree Search
Observed World $\phi(x_i w_i)$
$o_1 \leftarrow \pi^i_{\mathcal{O}}(\phi(x_i w_i))$
$o_2 \leftarrow \pi^i_{\mathcal{O}}(\phi(x_i w_i))$
World 1 $x^1_i w^1_i$ (Terminal)
World 2 $x^2_i w^2_i$
$o_3 \leftarrow \pi^i_{\mathcal{O}}(\phi(x^2_i w^2_i))$
World 3 $x^3_i w^3_i$ (Terminal)
Update Option Policy $\pi^{i+1}_{\mathcal{O}} \leftarrow \text{UpdatePolicy}(\pi^i_{\mathcal{O}})$

Figure 2: Workflow of the proposed system.

2.1 Planning Algorithm

In a dynamic environment with many actors and temporal constraints, decomposing the problem into reasoning over goals and trajectories separately is infeasible. Instead, we use learned policies together with an approach based on a variant of MCTS with two specializations. First, we replace the usual Upper Confidence Bound (UCB) weight with the term from [9] as follows:

$$Q(s_i, o_i) = Q^*(s_i, o_i) + C\frac{P(s_i, o_i)}{1 + N(s_i, o_i)}$$

where $Q^*(s_i, o_i)$ is the average value of option o_i from simulated play, $N(s_i, o_i)$ is the number of times option o_i was observed from s_i, $N(s_i)$ is the number of times s_i has been visited, and $P(s_i, o_i)$ is the predicted value of option o_i from state s_i. The goal of this term is to encourage useful exploration while focusing on option choices that performed well according to previous experience; it grants a high weight to any terms that have a high prior probability from our learned model.

Next, we use Progressive Widening to determine when to add a new node. This is a common approach for dealing with Monte Carlo tree search in a large search space [4], which functions by limiting the number of children added to each node in the search tree. Whenever we add a new node to the search tree, we use the current high-level policy to explore until we reach a termination condition.

After selecting an option to explore, we call the SIMULATE function to evolve the world forward in time. During this update, we check the full set of LTL constraints Φ and associated option constraints φ_o. If these are not satisfied, the search has arrived in a terminal node and a penalty is applied.

2.2 Model Checking

Each discrete option is associated with an LTL formula φ_o which establishes conditions that must hold while applying that option. We can evaluate $u_i = \pi_{\mathcal{U}}(o, \phi(x_i w_i))$ to get the next control as long as φ_o holds. In addition, we have a shared set Φ of LTL formulae that constrain the entire planning problem. At every step, we check whether a sampled trajectory satisfies all associated LTL formulae. Since we are checking satisfaction over finite runs, we use a bounded-time semantics for LTL [1]. We model check finite runs by leveraging the duality between LTL formulas and Deterministic Rabin Automata (DRAs). We precompute maximal accepting and rejecting strongly connected components of the DRA, which enables model checking in time linear in the length of the run.

3 Self Driving Car Domain

We apply our approach to the problem of planning for a self-driving car passing through an all-way stop intersection. To successfully complete the task the car must stop at a stop sign, wait until its turn to move, and accelerate back to the reference speed while avoiding collisions and changing lanes as necessary.

We break this down into a set of mid-level options: $\mathcal{O} = \{\texttt{Default}, \texttt{Follow}, \texttt{Pass}, \texttt{Stop}, \texttt{Wait}, \texttt{Left}, \texttt{Right}, \texttt{Finish}\}$, where the `Default` option represents driving down a straight road and stopping at a stop sign. The `Wait` option defines behavior at an intersection with multiple other vehicles. Other options such as `Stop` and `Finish` represent the agent's behavior on a crowded road before and after the intersection, respectively. We employ a common non-slip second-order nonholonomic model for vehicle dynamics that is often sufficient for realistic motion planning in nominal driving conditions.

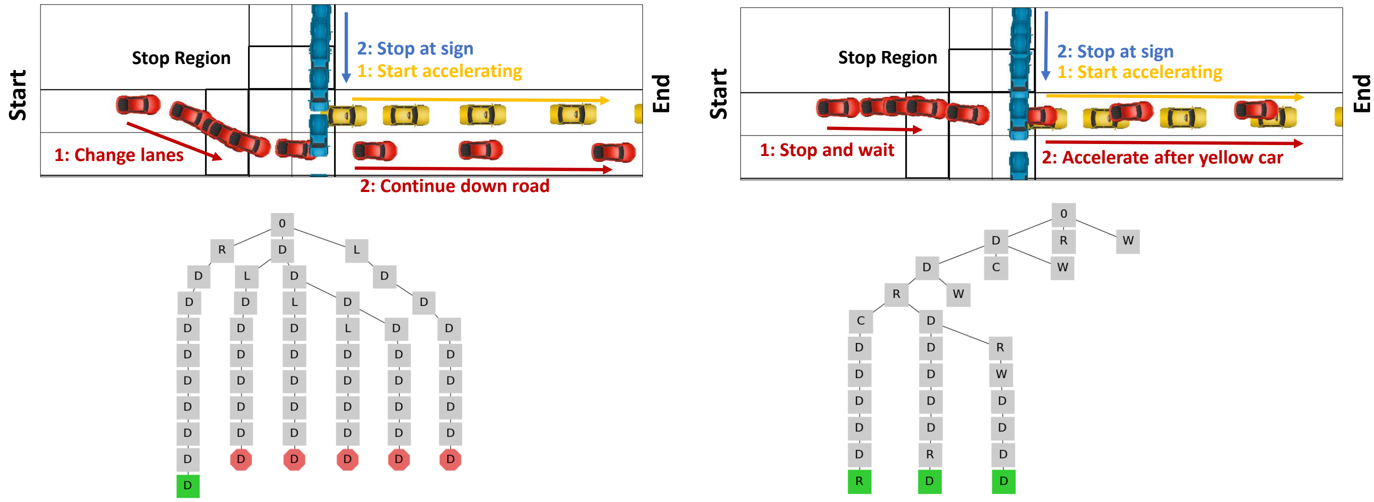


Figure 3: Comparison of MCTS on a test problem with a stopped car. Letters indicate option being executed: 0 = root, D = default "stay in lane" policy, W = wait, C = Finish/complete level, R = lane change to the right. On the left, we see tree search with a manually defined preference; on the right, we see the tree using the high-level policy acquired through DQN. Green leaves indicate success; red leaves indicate failure. The right side finds a higher-reward solution faster.

Our scenarios take place at the intersection of two two-lane, one-way roads. Each lane is 3 meters wide with a 25 mph speed limit, corresponding to common urban or suburban driving conditions. Stop signs are described as "stop regions": areas on the road that vehicles must come to a stop in before proceeding. Other vehicles follow a manually defined driving policy, designed for good performance under expected driving conditions.

3.1 Cost and Constraints

As described in Section 2.2, each discrete option is associated with an LTL formula φ_o which establishes conditions that must hold while applying it, and we also have a shared set Φ of LTL formulae constraining the entire plan. For example, for the entire road scenario, $\Phi =$

$$\square\,(\text{in\_stop\_region} \Rightarrow (\text{in\_stop\_region}\ \mathcal{U}\ \ \text{has\_stopped\_in\_stop\_region}))$$
$$\square((\text{in\_intersection} \Rightarrow \text{intersection\_is\_clear}) \wedge \neg\text{in\_intersection}\ \mathcal{U}\ \text{higher\_priority})$$

The reward function is a combination of a a cost term based on the current continuous state and a bonus based on completing intermediate goals or violating constraints (e.g. being rejected by the DRA corresponding to an LTL formula). The cost term penalizes the control inputs, acceleration and steering angle rate, as well as jerk, steering angle acceleration, and lateral acceleration. There are additional penalties for being far from the current reference speed and for offset from the center of the road. We add an additional penalty for being over the reference speed, set to discourage dangerous driving while allowing some exploration below the speed when interacting with other vehicles. We add terminal penalties for hitting obstacles or violating constraints. The other portion of the reward function is a penalty for constraint violations and a bonus for arriving at goals.

3.2 Learning

All control policies were represented as multilayer perceptrons with a single hidden layer of 32 fully connected neurons. We used the ReLu activation function on both the input and hidden layer, and the tanh activation function on the outputs. Outputs mapped to steering angle rate $\dot{\psi} \in [-1, 1]$ rad/s and acceleration $a \in [-2, 2]$ m/s^2. Control policies were trained according to the Deep Direct Policy Gradients algorithm [5]. We then performed Deep Q learning [6] on the discrete set of options to learn our high-level options policy. High-level policies were trained on a challenging road environment with 0 to 6 randomly placed cars with random velocity, plus a 50% chance of a stopped vehicle ahead in the current lane. We use Deep Q Learning as per [6] to learn a stochastic high-level policy over the set of options.

4 Results

We generated 100 random worlds in a new environment containing 0-5 other vehicles. We also test in the case with an additional stopped car in the lane ahead. Results are given in Table 1. For cases with the learned or simple action sets, we performed 100

Table 1: Comparison of different algorithms on 100 randomly-generated road environments with multiple other vehicles and with a stopped car on the road ahead.

Stopped Car	Low-Level	High-Level	Constraint Violations	Collisions	Total Failures	Avg Reward	Std Dev
Yes	Manual Policy	None	0	0	0	117.8	60.4
	Simple	Manual	0	5	7	105.5	93.0
	Simple	Learned	0	5	9	108.2	102.7
	256 NN Policy	None	3	4	6	109.4	95.7
	Learned	Manual	0	4	4	124.2	97.8
	Learned	Learned	0	0	0	137.8	74.2
No	Manual Policy	None	0	19	19	27.2	142.1
	Manual	Uniform	0	25	34	9.3	162.7
	Manual	Learned	1	27	36	7.4	163.2
	256 NN Policy	None	4	29	33	-9.1	149.3
	Learned	Manual	1	9	11	83.7	102.2
	Learned	Learned	0	3	3	95.2	74.2

iterations of MCTS to a time horizon of 10 seconds and select the best path to execute. Fig. 3 shows how the algorithm works in practice with different methods for choosing the high-level policy. With the learned high-level policy, we see excellent performance on the test set for simple problems and three failures in complex problems. These failures represent cases where there was a car moving at the same speed in the adjacent lane and a stopped car a short distance ahead, and there was no good option to explore. Our system avoids these situations where it is possible. When it predicts that such a situation will arise, our planner would give us roughly 2 seconds of warning to execute an emergency stop and avoid collision.

5 Conclusions

Our approach allows off-the-shelf DRL techniques to generalize to challenging new environments, and allows us to verify their behavior in these environments. However, there are several avenues for improvement. First, we did not focus on achieving high performance of the low-level policies, and the policies we learned had some instabilities. Second, we use termination conditions based on fixed time during tree search, but more complex conditions are possible. Additionally, choosing the best set of options is still an open problem and it is not clear what the best set of features is for this problem. In the future, we wish to extend this work to use stochastic control policies, and will also apply our system to real robots. For more details, see the full version of the paper [7].

References

[1] Armin Biere, Keijo Heljanko, Tommi A. Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5), 2006. doi: 10.2168/LMCS-2(5:5)2006. URL `http://dx.doi.org/10.2168/LMCS-2(5:5)2006`.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[3] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, 2001. ISBN 978-0-262-03270-4.

[4] Adrien Couetoux, Mario Milone, Matyas Brendel, Hassen Doghmen, Michele Sebag, Olivier Teytaud, et al. Continuous rapid action value estimates. In *The 3rd Asian Conference on Machine Learning (ACML2011)*, volume 20, pages 19–31. JMLR, 2011.

[5] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[7] Chris Paxton, Vasumathi Raman, Gregory D. Hager, and Marin Kobilarov. Combining neural networks and tree search for task and motion planning in challenging environments. *CoRR*, abs/1703.07887, 2017. URL `http://arxiv.org/abs/1703.07887`.

[8] Erion Plaku and Sertac Karaman. Motion planning with temporal-logic specifications: Progress and challenges. *AI Communications*, (Preprint):1–12.

[9] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[10] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence*, 2015.

[11] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. *arXiv preprint arXiv:1612.01079*, 2016.

Sufficient Markov Decision Processes with Alternating Deep Neural Networks

Longshaokan Wang
Department of Statistics
North Carolina State University
Raleigh, NC 27695
`lwang31@ncsu.edu`

Eric B. Laber
Department of Statistics
North Carolina State University
Raleigh, NC 27695
`eblaber@ncsu.edu`

Katie Witkiewitz
Department of Psychology
University of New Mexico
Albuquerque, NM 87106
`katiew@unm.edu`

Abstract

Advances in mobile computing technologies have made it possible to monitor and apply data-driven interventions across complex systems in real time. Markov decision processes (MDPs) are the primary model for sequential decision problems with a large or indefinite time horizon. Choosing a representation of the underlying decision process that is both Markov and low-dimensional is non-trivial. We propose a method for constructing a low-dimensional representation of the original decision process for which: 1. the MDP model holds; 2. a decision strategy that maximizes cumulative reward when applied to the low-dimensional representation also maximizes cumulative reward when applied to the original process. We use a deep neural network to define a class of potential process representations and estimate the process of lowest dimension within this class. The method is evaluated using a suite of simulation experiments, and applied to data from a mobile health intervention targeting smoking and heavy episodic drinking among college students.

Keywords: sufficient statistics, dimension reduction, deep learning

1 Introduction

Sequential decision problems arise in a wide range of application domains including autonomous vehicles, finance, robotics, and healthcare. Markov decision processes (MDPs) (Bellman, 1957; Puterman, 2014) are the primary mathematical model for representing sequential decision problems with an indefinite time horizon (Sutton and Barto, 1998). This class of models is quite general as almost any decision process can be made into an MDP by concatenating data over multiple decision points; however, coercing a decision process into the MDP framework in this way can lead to high-dimensional system state information that is difficult to model effectively. One common approach to construct a low-dimensional decision process from a high-dimensional MDP is to create a finite discretization of the space of possible system states and to treat the resultant process as a finite MDP. However, such discretization can result in a significant loss of information and can be difficult to apply when the system state information is continuous and high-dimensional. Another common approach to dimension reduction is to construct a low-dimensional summary of the underlying system states, e.g., by applying principal components analysis (Jolliffe, 1986), multidimensional scaling (Borg and Groenen, 1997), or by constructing a local linear embedding (Roweis and Saul, 2000). These approaches can identify a low-dimensional representation of the system state but, as we shall demonstrate, they need not retain salient features for making good decisions.

The preceding methods seek to construct a low-dimensional representation of a high-dimensional MDP with the goal of using the low-dimensional representation to estimate an optimal decision strategy, i.e., one that leads to maximal cumulative reward when applied to the original process; however, they offer no guarantee that the resulting process is an MDP or that a decision strategy estimated using data from the low-dimensional process will perform well when applied to the original process. We derive sufficient conditions under which a low-dimensional representation is an MDP, and that an optimal decision strategy for this low-dimensional representation is optimal for the original process. We develop a hypothesis test for this sufficient condition based on the Brownian distance covariance (Székely and Rizzo, 2009) and use this test as the basis for selecting a low-dimensional representation within a class of deep neural networks. The proposed estimator can be viewed as a novel variant of deep neural networks for feature construction in MDPs.

2 Sufficient Markov Decision Processes

Consider a discrete time stochastic control process: At time t, the process is in state $\mathbf{S}_t$. The agent/decision-maker executes action A_t. Then the process transitions to state $\mathbf{S}_{t+1}$ with transition probability $\mathcal{P}^{a_t}_{\mathbf{S}_t,\mathbf{S}_{t+1}} = \Pr(\mathbf{S}_{t+1} = \mathbf{s}_{t+1} \mid \mathbf{S}_t = \mathbf{s}_t, A_t = a_t)$, and provides a reward R_t with expectation $\mathcal{R}^{a_t}_{\mathbf{S}_t,\mathbf{S}_{t+1}} = \mathrm{E}[R_t \mid \mathbf{S}_t = \mathbf{s}_t, A_t = a_t, \mathbf{S}_{t+1} = \mathbf{s}_{t+1}]$. A Markov Decision Process is such a process that conditional on the current state and action, the reward and next state are independent from all previous states and actions. A deterministic policy π is a map from the state space $\mathcal{S}$ to the action space $\mathcal{A} : \pi(s_t) = a_t$, and defines an agent's behavior. The value of a state $\mathbf{s}$ under a certain policy π is the expected cumulative reward if the current state is s and the agent follows π for action selection: $V^\pi(\mathbf{s}) = \mathrm{E}_\pi\{\sum_{k=0}^\infty \gamma^k R_{t+k} \mid \mathbf{S}_t = \mathbf{s}\}$, where the discount factor $\gamma \in [0, 1)$ determines the importance of short-term rewards compared to long-term rewards. Similarly, the value of a state-action pair, referred to as the Q-function, is the expected cumulative reward if the current state is $\mathbf{s}$, current action is a, and the agent follows π afterwards: $Q^\pi(\mathbf{s}, a) = \mathrm{E}_\pi\{\sum_{k=0}^\infty \gamma^k R_{t+k} \mid \mathbf{S}_t = \mathbf{s}, A_t = a\}$. An optimal policy π^* yields the largest value for every state or state-action pair. Denote the corresponding Q-function as $Q^*(\mathbf{s}, a) = Q^{\pi^*}(\mathbf{s}, a) = \max_\pi Q^\pi(\mathbf{s}, a)$. If $Q^*(\mathbf{s}, a)$ is known for all $\mathbf{s} \in \mathbf{S}$ and $a \in \mathcal{A}$, then an optimal policy is simply $\pi^*(\mathbf{s}) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(\mathbf{s}, a)$. Q^* satisfies the Bellman Optimality Equations:

$$Q^*(\mathbf{s}, a) = \mathrm{E}\{R_t + \gamma \max_{a'} Q^*(\mathbf{S}_{t+1}, a') \mid \mathbf{S}_t = \mathbf{s}, a_t = a\} \quad \forall\, \mathbf{s} \in \mathbf{S}, a \in \mathcal{A}.$$

Now we are ready to introduce the criterion that guarantees a dimension reduction ϕ to retain all the relevant information for learning an optimal policy, by showing that under this criterion, the optimal policy learned from the dimension reduced states is the same as the optimal policy learned from the original states.

Theorem 2.1. *Let* $\{(\mathbf{S}_t, A_t, R_t)\}_{t=1,2,\ldots}$ *be an MDP where* $\mathbf{S}_t \in \mathbb{R}^p \;\; \forall\, t$. *Suppose* $\exists\, \phi : \mathbb{R}^p \to \mathbb{R}^q$ *where* $q < p$ *such that*

$$\{R_t, \mathbf{S}_{t+1} \perp\!\!\!\perp \mathbf{S}_t\} \mid \phi(\mathbf{S}_t), A_t, \tag{C1}$$

then $Q^*(\mathbf{s}, a) = \widetilde{Q}^*(\phi(\mathbf{s}), a)$ *and* $\pi^*(\mathbf{s}) = \widetilde{\pi}^*(\phi(\mathbf{s}))$.

Theorem 2.1 provides a general framework regardless of the implementations of testing for C1 and constructing $\widehat{\phi}$ in practice. Next we will discuss a procedure to check for C1 given $\widehat{\phi}$, while leaving our proposed $\widehat{\phi}$ to Section 3.

Checking for C1 requires a measure of arbitrary conditional dependence. Common approaches like partial correlation only measure linear conditional dependence. Székely and Rizzo (2009) showed that distance correlation (dCor) and distance covariance test (dCov test) can measure and test for arbitrary dependence. Wang et al. (2015) extended dCor and dCov test to conditional distance correlation (CDCor) and conditional independence test (CDIT), but CDIT requires bandwidth selection and is not very powerful from our experiments. Given the difficulties with testing for conditonal dependence, we propose to replace a direct test for C1 with tests for dependence without conditioning on $\widehat{\phi}(\mathbf{S}_t)$, which would allow us to use the fast and relatively powerful dCov test when $\mathbf{S}_t$ and R_t are continuous variables. The following lemmas provide sufficient conditions for C1. For notational simplicity let $(\mathbf{S}R)_{t+1} = (\mathbf{S}_{t+1}, R_t)$, the concatenation.

Lemma 2.2. *If*

$$\{(\mathbf{S}R)_{t+1} - E[(\mathbf{S}R)_{t+1} \,|\, \phi(\mathbf{S}_t), A_t] \perp\!\!\!\perp \mathbf{S}_t\} \,|\, A_t, \tag{C2}$$

then C1 holds.

Thus, instead of checking for C1, one can check for C2 within levels of A^t. In practice, first select the corresponding $\mathbf{s}_t, r_t$, and $\mathbf{s}_{t+1}$ for each a_t. Next, compute $\widehat{\phi}(\mathbf{s}_t)$. Lastly, estimate $\widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \widehat{\phi}(\mathbf{S}_t) = \widehat{\phi}(\mathbf{s}_t), A_t = a_t]$ and use dCov test on $(\mathbf{s}r)_{t+1} - \widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \widehat{\phi}(\mathbf{S}_t) = \widehat{\phi}(\mathbf{s}_t), A_t = a_t]$ and $\mathbf{s}_t$. If dCov test shows independence for each a_t, then $\widehat{\phi}$ satisfies C1 and is a sufficient dimension reduction.

We end this section with a discussion on what properties $\widehat{\phi}$ should have. C1 suggests that $\widehat{\phi}$ needs to retain the information in $\mathbf{S}_t$ relevant to $(\mathbf{S}R)_{t+1}$, unlike common dimension reduction techniques which typically aim to retain the information about $\mathbf{S}_t$ itself. Therefore, it seems the most reasonable approach to perform supervised learning and build a model that predicts $(\mathbf{S}R)_{t+1}$ from $\mathbf{S}_t$ and A_t, and obtain $\widehat{\phi}$ from the predictive model. For example, we might consider using artificial neural networks (ANN) whose input is $\mathbf{s}_t$ and a_t and output is $\widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \mathbf{S}_t = \mathbf{s}_t, A_t = a_t]$, and let $\widehat{\phi}(\mathbf{s}_t)$ be the the output from a hidden layer with lower dimension than s_t. However, because eventually we wish to apply the estimated optimal policy to new data and use $a_t = \operatorname{argmax}_a \widehat{Q}^*(\widehat{\phi}(\mathbf{s}_t), a)$ for action selection, $\widehat{\phi}(\mathbf{s}_t)$ cannot depend on a_t, but instead contains the union of relevant information with respect to $(\mathbf{S}R)_{t+1}$ for all possible a_t. Therefore, in the ANN example, we cannot add a_t to the input layer, or build a separate ANN model for each a_t. On the other hand, the output $\widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \mathbf{S}_t = \mathbf{s}_t, A_t = a_t]$ still depends on a_t. This dilemma inspires us to design a new deep learning algorithm to construct $\widehat{\phi}$.

3 Alternating Deep Neural Networks

The algorithm we propose for constructing a sufficient dimension reduction is based on deep neural networks (DNN), which is essentially artificial neural networks (ANN) with multiple hidden layers and various improvements (LeCun et al., 2015). A DNN model is usually composed of an input layer, multiple hidden layers, and an output layer. Data is propagated forward through the network via linear transformations between layers followed by possibly non-linear activation functions within each layer to obtain the output. More precisely, for the l-th layer, let $\mathbf{o}^{l-1} \in \mathbb{R}^{p\times 1}$ be the input, $\mathbf{w}^l \in \mathbb{R}^{q\times p}$ be the weight matrix, $\mathbf{b}^l \in \mathbb{R}^{q\times 1}$ be the bias vector, φ^l be the element-wise activation function which is typically sigmoid or rectified linear unit (ReLU), and $\mathbf{o}^l \in \mathbb{R}^{q\times 1}$ be the output, then $\mathbf{o}^l = \varphi^l(\mathbf{w}^l \cdot \mathbf{o}^{l-1} + \mathbf{b}^l)$. Let $\mathbf{x}_i = \mathbf{o}_i^1$ be the i-th initial input, $\widehat{\mathbf{y}}_i = \mathbf{o}_i^L$ be the i-th final output, $\boldsymbol{\theta}$ be the set of all parameters. The network is trained to minimize a cost function, typically the root mean squared error (MSE) of predictions: $C(\boldsymbol{\theta}) = \sqrt{\frac{1}{N}\sum_{i=1}^N \|\mathbf{y}_i - \widehat{\mathbf{y}}_i\|_2^2}$ .The gradient of the cost function is propagated backwards through the network to update all the parameters using Stochastic Gradient Descent (SGD) or its variations.

We add a new structure and training procedure to DNN to construct the dimension reduction function $\widehat{\phi}$, so that we can solve the dilemma that $\widehat{\phi}$ doesn't depend on a_t while the prediction $\widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \mathbf{S}_t = \mathbf{s}_t, A_t = a_t]$ does. Our algorithm, which we call Alternating Deep Neural Networks (ADNN), uses

- $\mathbf{s}_t$ as the input,
- $(\mathbf{s}r)_{t+1}$ as the response,
- root MSE of $\widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \mathbf{S}_t = \mathbf{s}_t, A_t = a_t]$ as the cost,
- a dimension reduction network that computes $\widehat{\phi}(\mathbf{s}_t)$ independent of a_t as the first half of ADNN,
- a prediction network that computes $\widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \widehat{\phi}(\mathbf{S}_t) = \widehat{\phi}(\mathbf{s}_t), A_t = a_t]$ as the second half of ADNN.

The dimension reduction network is just a DNN whose last layer has the dimension of $\widehat{\phi}(\mathbf{S}_t)$. On the other hand, the prediction network is multiple branches of DNN, each branch corresponding to a particular a_t. The dimension reduction network is connected to the prediction network in a decision-tree fashion: $\mathbf{s}_t$ propagates through the dimension reduction network to obtain $\widehat{\phi}(\mathbf{s}_t)$, during which a_t is irrelevant, then $\widehat{\phi}(\mathbf{s}_t)$ propagates through the branch corresponding to a_t in the prediction network to obtain $\widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \widehat{\phi}(\mathbf{S}_t) = \widehat{\phi}(\mathbf{s}_t), A_t = a_t]$ (Figure 1). Except for having the same input,

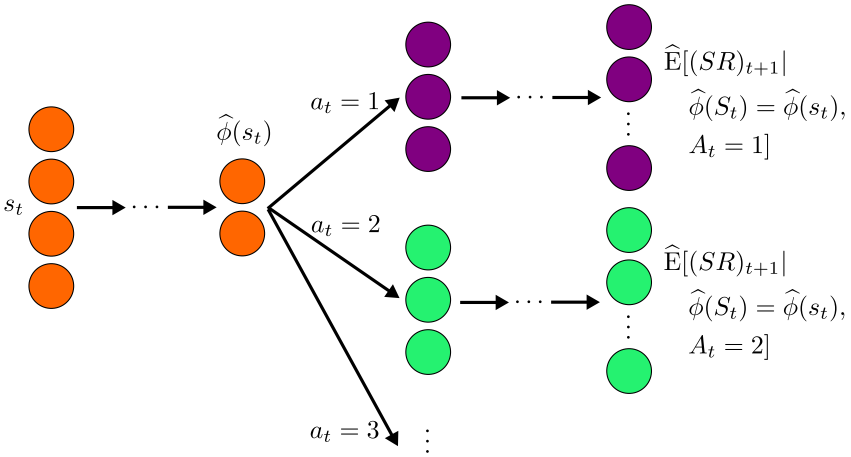

s_t
$\widehat{\phi}(s_t)$
$a_t = 1$
$a_t = 2$
$a_t = 3$
$\widehat{\mathrm{E}}[(SR)_{t+1}|$
$\widehat{\phi}(S_t) = \widehat{\phi}(s_t),$
$A_t = 1]$
$\widehat{\mathrm{E}}[(SR)_{t+1}|$
$\widehat{\phi}(S_t) = \widehat{\phi}(s_t),$
$A_t = 2]$

Figure 1: Structure of ADNN

the multiple branches are unrelated: They could have distinct parameters and even structures, allowing the prediction network to uncover possibly different relationships between $\widehat{\phi}(\mathbf{S}_t)$ and $(\mathbf{S}R)_{t+1}$ for each a_t.

Note that in ADNN, $\widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \mathbf{S}_t = \mathbf{s}_t, A_t = a_t] = \widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \widehat{\phi}(\mathbf{S}_t) = \widehat{\phi}(\mathbf{s}_t), A_t = a_t]$ by design, because for a true sufficient dimension reduction ϕ, $\mathrm{E}[(\mathbf{S}R)_{t+1} \,|\, \mathbf{S}_t, A_t] = \mathrm{E}[(\mathbf{S}R)_{t+1} \,|\, \phi(\mathbf{S}_t), A_t]$. If $\widehat{\phi}$ is insufficient and fails to capture the useful information in $\mathbf{S}_t$ related to $(\mathbf{S}R)_{t+1}$ for various a_t, $\widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \widehat{\phi}(\mathbf{S}_t) = \widehat{\phi}(\mathbf{s}_t), A_t = a_t]$ would be a poor estimate for $\mathrm{E}[(\mathbf{S}R)_{t+1} \,|\, \mathbf{S}_t = \mathbf{s}_t, A_t = a_t]$ and the cost would be high. Thus we expect to learn a good approximation of ϕ by minimizing the cost, which can be written as $C(\boldsymbol{\theta}) = \sqrt{\frac{1}{NT}\sum_{t=1}^{T-1}\sum_{i=1}^{N}\left\|(\mathbf{s}r)_{t+1,i} - \widehat{\mathrm{E}}[(\mathbf{S}R)_{t+1} \,|\, \widehat{\phi}(\mathbf{S}_t) = \widehat{\phi}(\mathbf{s}_{t,i}), A_t = a_{t,i}]\right\|_2^2}$, where $\mathbf{s}_{t,i}$ is the state of subject i at time t and so on. We omit the subscript for subject when there is no confusion.

The training process of ADNN is also different from typical DNN. Suppose that $\mathcal{A} = \{1, 2, \cdots, k\}$. We first divide the data, $\{\mathbf{s}_{t,i}, \mathbf{s}_{t+1,i}, r_{t,i}, a_{t,i}\}_{t=1,\cdots,T-1}^{i=1,\cdots,N}$, into k action-categories based on $a_{t,i}$, one for each available action. In each iteration, we draw a random batch alternately from every action-category to perform SGD, hence the name "Alternating Deep Neural Network". We allow the dimension reduction network to learn the union of information in $\mathbf{S}_t$ that is important to $(\mathbf{S}R)_{t+1}$ for all $a \in \mathcal{A}$ via this alternating training algorithm.

We can enable ADNN to perform variable selection as part of the dimension reduction, by adding a group lasso penalty to the columns of the weight matrix of the first hidden layer. The j-th input variable is deleted if the j-th column of $\mathbf{w}^1$ is a vector of 0's. Variables that are deleted don't need to be measured and stored in future data collection, reducing the financial cost in cases like clinical trials, in which measuring patient covariates can be expensive.

4 Simulation Experiments

We evaluate the finite sample performance of ADNN using a series of simulation experiments. To form a basis for comparison, we consider two alternative feature construction methods: (PCA) principal component analysis, so that the estimated feature map $\widehat{\phi}_{\mathrm{PCA}}(\mathbf{s})$ is the projection of $\mathbf{s}$ onto the first k principal components; and (tDNN) a "traditional" deep neural network, which can be seen as a special case of ADNN where there is only 1 action. In our implementation of PCA, we choose the number of principal components, k, corresponding to 90% of variance explained. In our implementation of tDNN, we build a separate tDNN for each $a \in \mathcal{A}$, where hyperparameters are tuned using cross-validation, and take the union of selected variables and constructed features. Note that there is no other obvious way to join the constructed features from tDNN but to simply concatenate them, which will lead to inefficient dimension reduction especially when $|\mathcal{A}|$ is large, whereas we will see that ADNN provides a much more efficient way to aggregate the useful information across actions.

We evaluate the quality of a feature map, ϕ, in terms of the marginal mean reward in test MDPs under the estimated optimal regime constructed from the training MDPs using Q-learning with function approximation; we use both linear function approximation and non-linear function approximation with neural networks.

We consider data from the following class of generative models, where superscripts indicate components:

$$\begin{aligned}
&\mathbf{S}_1 \sim \text{Normal}_{64}\left(0,\ 0.25\mathbf{I}_{64}\right), \quad A_1, \ldots, A_T \sim_{i.i.d.} \text{Bernoulli}\left(0.5\right), \\
&S_{t+1}^{4i-3}, S_{t+1}^{4i-2} \sim_{i.i.d.} \text{Normal}\left\{(1-A_t)g(S_t^i),\ 0.01(1-A_t) + 0.25A_t\right\}, \\
&S_{t+1}^{4i-1}, S_{t+1}^{4i} \sim_{i.i.d.} \text{Normal}\left\{A_t g(S_t^i),\ 0.01A_t + 0.25(1-A_t)\right\}, \\
&R_t \sim \text{Normal}\{(1-A_t)[2\{g(S_t^1) + g(S_t^2)\} - \{g(S_t^3) + g(S_t^4)\}] \\
&\qquad\qquad + A^t[2\{g(S_t^3) + g(S_t^4)\} - \{g(S_t^1) + g(S_t^2)\}],\ 0.01\}, \\
&\text{for } i = 1, 2, \ldots, 16.
\end{aligned}$$

Note that the relationship between $\mathbf{S}_t$ and $(\mathbf{S}R)_{t+1}$ depends on the action. The first 16 variables determine the next state, and the first 4 variables determine the reward.

The above class of models is indexed by $g : \mathbb{R} \to \mathbb{R}$ which we vary across the following maps: identity $g(u) = u$, truncated quadratic $g(u) = \min\{u^2, 3\}$, and truncated exponential $g(u) = \min\{\exp(u), 3\}$, where the truncation is used to keep all variables of relatively the same scale across time points. Additionally, we add 3 types of noise variables, each taking up about $\frac{1}{3}$ of total noises added: (i) dependent noise variables, which are generated the same way as above except that they don't affect the reward; (ii) white noises, which are sampled independently from $\text{Normal}(0, 0.25)$ at each time point; and (iii) constants, which are sampled independently from $\text{Normal}(0, 0.25)$ at $t = 1$ and remain constant over time. It can be seen that the first 16 variables, the first 4 variables, and $\{g(S_t^1), g(S_t^2), g(S_t^3) + g(S_t^4)\}^\intercal$ all induce a sufficient MDP. the foregoing class of models is designed to evaluate the ability of the proposed method to identify low-dimensional and potentially nonlinear features of the state in the presence of action-dependent transitions and various noises. For each Monte Carlo replication, we sample $n = 30$ i.i.d. trajectories of length $T = 90$ from the above generative model.

The results based on 500 Monte Carlo replications are reported in Table 1. (Only the quadratic model is shown due to space limit. Results for linear and exponential models are similar.) In addition to reporting the mean reward under the policy estimated using Q-learning with both function approximations, we also report: (nVar) the number of selected variables; and (nDim) the dimension of the feature map. The table shows that (i) ADNN produces significantly smaller nVar and nDim compared with PCA or tDNN in all cases; (ii) ADNN is robust to the 3 types of noises; (iii) when fed into the Q-learning algorithm, ADNN leads to consistently greater marginal mean reward than PCA, and greater reward than the original states under non-linear models; and (iv) ADNN is able to construct features suitable for Q-learning with linear function approximation even when the reward function and transition between states are non-linear.

Model	nNoise	Feature map	Linear Q	NNQ	nVar	nDim
quad	0	$\mathbf{s}_t$	3.05(0.068)	3.74(0.064)	64	64
		$(s_t^1, s_t^2, s_t^3, s_t^4)^\intercal$	2.68(0.057)	5.57(0.043)	4	4
		$\widehat{\phi}_{\text{ADNN}}(\mathbf{s}_t)$	6.69(0.030)	7.13(0.030)	4.1(0.02)	2.4(0.04)
		$\widehat{\phi}_{\text{tDNN}}(\mathbf{s}_t)$	7.45(0.029)	6.71(0.067)	15.3(0.04)	37.1(0.22)
		$\widehat{\phi}_{\text{PCA}}(\mathbf{s}_t)$	2.77(0.056)	2.49(0.056)	64	51.2(0.02)
	50	$\mathbf{s}_t$	2.49(0.044)	3.21(0.053)	114	114
		$(s_t^1, s_t^2, s_t^3, s_t^4)^\intercal$	1.90(0.039)	6.32(0.057)	4	4
		$\widehat{\phi}_{\text{ADNN}}(\mathbf{s}_t)$	6.65(0.027)	6.98(0.023)	6.4(0.06)	5.3(0.09)
		$\widehat{\phi}_{\text{tDNN}}(\mathbf{s}_t)$	6.90(0.026)	6.33(0.060)	36.5(0.03)	88.3(0.22)
		$\widehat{\phi}_{\text{PCA}}(\mathbf{s}_t)$	3.13(0.059)	3.02(0.061)	114	87.1(0.03)
	200	$\mathbf{s}_t$	1.22(0.030)	1.29(0.054)	264	264
		$(s_t^1, s_t^2, s_t^3, s_t^4)^\intercal$	2.30(0.038)	5.65(0.038)	4	4
		$\widehat{\phi}_{\text{ADNN}}(\mathbf{s}_t)$	6.89(0.028)	6.92(0.031)	14.3(0.14)	12.5(0.23)
		$\widehat{\phi}_{\text{tDNN}}(\mathbf{s}_t)$	6.76(0.043)	6.41(0.065)	84.1(0.11)	152.4(0.36)
		$\widehat{\phi}_{\text{PCA}}(\mathbf{s}_t)$	3.24(0.062)	1.81(0.074)	264	167.4(0.03)

Table 1: Comparison of feature map estimators under linear and different number of noise variables (nNoise) in terms of: marginal mean reward using Q-learning with linear function approximation (Linear Q); marginal mean reward using Q-learning with neural network function approximation (NN Q); the number of selected variables (nVar); and the dimension of the feature map (nDim)

Deep and Shallow Approximate Dynamic Programming

Nir Levine
Department of Electrical Engineering
The Technion
Haifa, Israel 32000
`levinir@campus.technion.ac.il`

Tom Zahavy
Department of Electrical Engineering
The Technion
Haifa, Israel 32000
`tomzahavy@campus.technion.ac.il`

Daniel J. Mankowitz
Department of Electrical Engineering
The Technion
Haifa, Israel 32000
`danielm@tx.technion.ac.il`

Aviv Tamar
Dept. of Electrical Engineering and Computer Sciences
UC Berkeley
Berkeley, CA 94720
`avivt@berkeley.edu`

Shie Mannor
Department of Electrical Engineering
The Technion
Haifa, Israel 32000
`shie@ee.technion.ac.il`

Abstract

Deep Reinforcement Learning (DRL) agents have achieved state-of-the-art results in a variety of challenging, high-dimensional domains. This success is mainly attributed to the power of Deep Neural Networks to learn rich domain representations while approximating the value function or policy end-to-end. However, DRL algorithms are non-linear temporal-difference learning algorithms, and as such, do not come with convergence guarantees and suffer from stability issues. On the other hand, linear function approximation methods, from the family of Shallow Approximate Dynamic Programming (S-ADP) algorithms, are more stable and have strong convergence guarantees. These algorithms are also easy to train, yet often require significant feature engineering to achieve good results. We utilize the rich feature representations learned by DRL algorithms and the stability and convergence guarantees of S-ADP algorithms, by unifying these two paradigms into a single framework. More specifically, we explore unifying the Deep Q Network (DQN) with Least Squares Temporal Difference Q-learning (LSTD-Q). We do this by re-training the last hidden layer of the DQN with the LSTD-Q algorithm. We demonstrate that our method, LSTD-Q Net, outperforms DQN in the Atari game Breakout and results in a more stable training regime.

Keywords: Reinforcement Learning, Deep Reinforcement Learning, Approximate Dynamic Programming, Deep Q Network, Least Squares Temporal Difference

1 Introduction

Reinforcement Learning (RL) is a field of research that uses Dynamic Programing (DP), among other approaches, to solve sequential decision making problems. The main challenge in applying DP to real world problems is the exponential growth of the computational and storage requirements as the number of state variables and control variables increases, also known as the curse of dimensionality.

Approximate DP (ADP) tries to tackle this problem by approximating the value function, e.g., with a linear function approximator (Shallow-ADP (S-ADP)) or a Deep Neural Network (DNN). Generally speaking, linear architectures are easier to train, whereas non-linear ones (e.g., the Deep Q Network (DQN, (Mnih et al., 2015))) learn richer domain representations (Wang et al., 2015; Zahavy et al., 2016) and achieve superior performance. We are interested in an ADP algorithm with the following properties: ***(1)*** Achieves **high performance** in multiple domains without domain specific knowledge; ***(2)*** **Stability** - has low bias and variance; ***(3)*** **Safety** - enjoys theoretical convergence guarantees. ***(4)*** **Efficiency** - efficiently utilizes the data.

S-ADP methods have enjoyed wide popularity over the years (Tsitsiklis et al., 1997). These algorithms are known to be safe, stable and efficient. In particular, batch algorithms such as Least Squares Temporal Difference (LSTD, Lagoudakis & Parr (2003)) are known to be more data efficient than gradient-based and online temporal difference approaches. However, the success of these algorithms crucially depends on the quality of the state representation. Ideally, the features are able to encode rich, possibly non-linear representations that are necessary for a given domain. However, in practice, finding good features is difficult and often hampers the usage of linear function approximation methods.

On the other hand, recent advancements in Deep Reinforcement Learning (DRL) have had highly successful results in solving a wide range of high dimensional input domains (e.g., Mujoco (Todorov et al., 2012), Atari (Bellemare et al., 2013) and Minecraft (Tessler et al., 2016)). DRL algorithms use a DNN to approximate the Q-function (or the policy) and learn features together with the value function end-to-end. While they achieve good performance in many domains, DRL algorithms suffer from high variance, biased gradients, divergence issues and require large amounts of data to converge. These issues are currently being addressed by the DRL community (e.g., Schulman et al. (2015); Munos et al. (2016)).

In this work, we develop an approach that aims to **unify** S-ADP algorithms and DRL (Figure 1). Our approach is to leverage the ability of DRL algorithms to learn rich representations, with the stability, efficiency and safety that characterize S-ADP methods. In particular, we examine the combination of a DQN algorithm with the LSTD-Q (Lagoudakis & Parr, 2003) algorithm. We show that by relearning the *last hidden layer* of a DQN with the LSTD-Q algorithm, we can train a policy that is more stable and achieves better performance than the DQN baseline in the Atari game Breakout.

It should be noted that, although we focus on combining the DQN and LSTD-Q algorithms, our framework is not limited to this specific unification scheme. We see our framework as providing a general approach to combining DRL with S-ADP algorithms.

Related Work: Using the hidden layers of a DNN as a feature extractor and learning the last layer with a different algorithm has been addressed before in supervised learning context (e.g. Donahue et al. (2013)). In an RL context, there have been competitive attempts to use S-ADP methods and unsupervised features to play Atari (Liang et al., 2016; Blundell et al., 2016). However, we are not familiar with any unifications approaches that combine S-ADP algorithms and DRL methods.

2 Background

Reinforcement Learning (RL) (Sutton & Barto, 1998): A Markov Decision Process (MDP) is defined as a five-tuple $\langle S, A, R, P, \gamma \rangle$. Here, S is a finite set of states, A a finite set of actions and $\gamma \in [0, 1]$ is the discount factor. A transition probability function $P : S \times A \to \Delta_S$ is a function mapping from states and actions to a probability distribution over next states. Finally, $R : S \times A \to [R_{min}, R_{max}]$ is a bounded reward function. The goal in RL is for an agent to learn a policy $\pi : S \to \Delta_A$ that solves the MDP by maximizing the expected discounted return. The action value function $Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_t = s, a_t = a, \pi\right]$ represents the expected discounted return of executing action $a \in A$ from state $s \in S$ and following the policy π thereafter. The optimal action-value function $Q^*(s, a)$ obeys a fundamental recursion known as the Bellman equation which is defined as $Q^*(s_t, a_t) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')\right]$.

Deep Q Networks (DQN) (Mnih et al., 2015): The DQN algorithm approximates the optimal Q function using a Convolutional Neural Network (CNN). The algorithm learns the network weights by minimizing the mean squared error of the Bellman equation defined as $\mathbb{E}_{s_t, a_t, r_t, s_{t+1}} \|Q_\theta(s_t, a_t) - y_t\|_2^2$, where $y_t = r_t$ if s_{t+1} is terminal, otherwise $y_t = r_t + \gamma \underset{a'}{\max} Q_{\theta_{target}}\left(s_{t+1}, a'\right)$. As can be seen in the equations, there are two sets of weights θ and θ_{target}. The

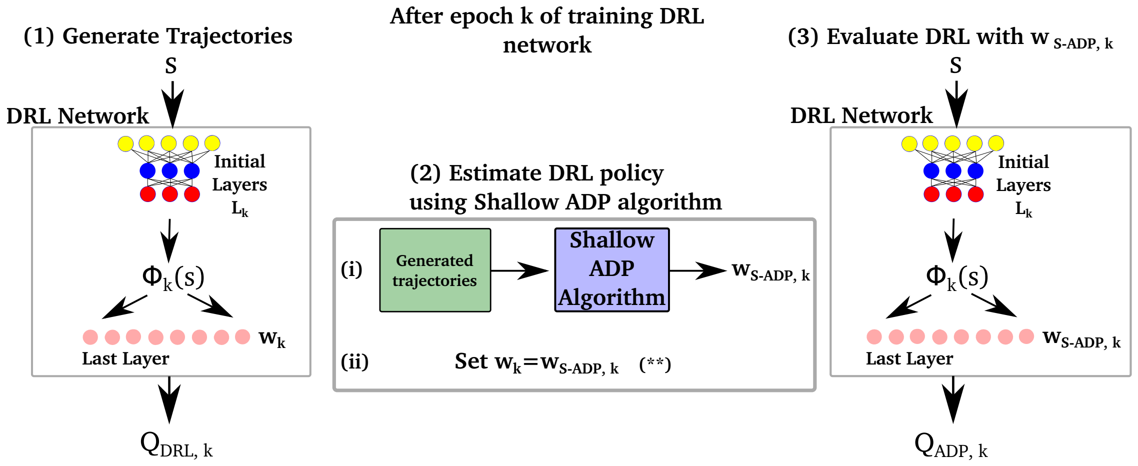


Figure 1: An overview of our approach that unifies DRL and S-ADP algorithms.

DQN maintains two separate networks, namely the current network with weights θ and the target network with weights θ_{target}. The weights of the target network are set to θ every fixed number of iterations. The DQN is an offline learning algorithm. Therefore, the tuples $\langle s_t, a_t, r_t, s_{t+1}\rangle$ that are used to optimize the network weights are first collected from the agent's experience and are stored in an Experience Replay (ER) buffer (Lin, 1993). When performing an optimization step, a mini-batch of samples are randomly selected from the buffer and are used to calculate the gradients. It is the addition of the target network and ER that significantly improved the stability and performance of the DQN.

Least Squares Temporal Different Q-Learning (LSTD-Q) (Lagoudakis & Parr, 2003): LSTD (Barto & Crites, 1996) and LSTD-Q come from the family of S-ADP (Bertsekas, 2008) algorithms. LSTD is an RL algorithm that estimates the state value function $V^{\pi}(s)$ from a batch of offline samples generated from a policy π with convergence guarantees. LSTD-Q extends LSTD to learning a control policy by estimating the action value function $Q^{\pi}(s,a)$. Let $Q^{\pi} \in \mathbb{R}^{|S||A|}$ be the unknown action-value function of a policy π. Let $\hat{Q}^{\pi} \in \mathbb{R}^{|S||A|}$ be an approximation to Q^{π} defined as $\hat{Q}^{\pi} = \Phi w^{\pi}$ where $w^{\pi} \in \mathbb{R}^{k}$ are a set of weights and $\Phi \in \mathbb{R}^{|S||A| \times k}$ is a feature matrix. Each row of Φ represents a feature vector for a certain pair $\langle s, a\rangle$. The goal is to learn the set of weights w^{π} by forcing the approximate value function to be a fixed point under the Bellman operator T^{π}. That is, $\hat{Q}^{\pi} \approx T^{\pi}\hat{Q}^{\pi}$. A simple derivation leads to computing the exact values of w^{π} by solving the linear $k \times k$ system $Aw^{\pi} = b$ where $A = \Phi^{T}(\Phi - \gamma \mathbf{P}\Pi_{\pi}\Phi)$ and $b = \Phi^{T}\mathcal{R}$. Here, $\mathcal{R} \in \mathbb{R}^{|S||A|}$ is the reward vector, $\mathbf{P} \in \mathbb{R}^{|S||A| \times |S|}$ is the transition matrix and $\Pi_{\pi} \in \mathbb{R}^{|S| \times |S||A|}$ is a matrix describing the policy. Given a set of samples $D = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}^{M}$, we can approximate A and b with the following update rules:

$$\tilde{A} = \frac{1}{M}\sum_{i=0}^{M}\left[\phi(s_i,a_i)\Big(\phi(s_i,a_i) - \gamma\phi(s_{i+1},\pi(s_{i+1}))\Big)^{T}\right] \tag{1}$$

$$\tilde{b} = \frac{1}{M}\sum_{i=1}^{M}\left[\phi(s_i,a_i)r_i\right] , \tag{2}$$

which in the limit converge to the true A and b. The weights w^{π} can be calculated using a least squares minimization:

$$\tilde{w}^{\pi} = \arg\min_{w} \|\tilde{A}w - \tilde{b}\|_2^2$$

or by calculating the pseudo-inverse: $\tilde{w}^{\pi} = \tilde{A}^{-1}\tilde{b}$. LSTD-Q is also sample efficient since the *same* set of samples D can be used to train any policy π so long as $\pi(s_{t+1})$ is available for every s_{t+1} in the set.

3 Algorithm

The main idea in our algorithm is to replace the policy that was learned by DRL techniques with one that is learned using S-ADP techniques (Figure 1). This is done at the end of every epoch in the following manner: **(1)** Generate trajectories using a simulator or by interacting with the environment. Translate these trajectories into tuples using $\Phi(\cdot,\cdot)$. In our case, $\Phi(\cdot)$ is defined to be the output of the last hidden layer in the DRL network. **(2)** Use the tuples and an S-ADP algorithm to estimate the DRL policy. In our case, the policy is defined to be the last layer in the DRL network. Therefore, this step involves estimating the weights of the last hidden layer, which we denote by $w_{\text{S-ADP},k}$. **(3)** Evaluate the policy obtained in (2) by replicating the DRL network from (1) and substitute its last layer weights w_k (i.e., the policy) with $w_{\text{S-ADP},k}$.

It is possible to continue training the DRL with the new weights $w_{\text{S-ADP},k}$ (see (**) step (2) of Figure 1). However, in this work we only evaluate the network's performance with this new policy, to show the potential benefits of this approach.

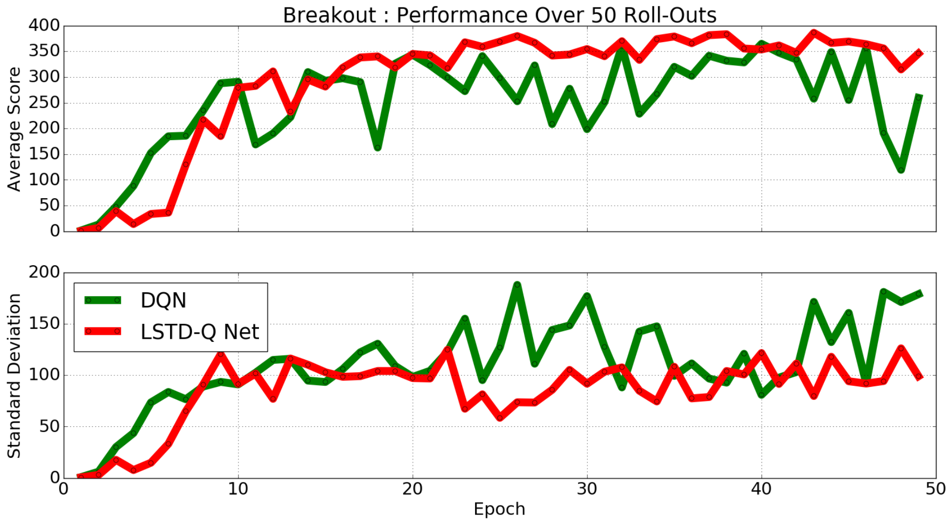


Figure 2: Average scores and standard deviations over 50 roll-outs.

4 Experiments

Setup: In our experiments, we used the DQN as the DRL network and LSTD-Q as the S-ADP algorithm. We trained a DQN on the the Atari2600 environment game called Breakout (Bellemare et al., 2013), for 50 million steps. After every one million steps, referred to as one epoch, we first stored a 'snapshot' of the network. The snapshot, following the notation in Figure 1, stored L_k and w_k for future evaluation. We then performed LSTD-Q to learn the weights $w_{\text{S-ADP},k}$ of the last layer (policy), using the data from the DQN snapshot's ER buffer (Discussed in more detail below). We stored both the original policy of the DQN snapshot w_k and the learned LSTD-Q policy $w_{\text{S-ADP},k}$ at each epoch.

Stability: When performing the experiments we noticed that the LSTD-Q algorithm lacked stability, due to $\tilde{A}$ being ill-posed. In order to stabilize the results, we added regularization in the form of additive terms to $\tilde{A}$'s diagonal. Specifically we solved $w_{\text{S-ADP},k} = \text{argmin}_w \, \|(\tilde{A} + \lambda)w - \tilde{b}\|_2^2$, where I is the identity matrix, and $\lambda = 5e-4$.

Data: The way in which data is generated to learn a policy using LSTD-Q has a significant impact on the performance of the algorithm. We initially generated data using the current snapshot of the DQN (i.e., L_k and w_k) simply by playing according to the current policy and collecting the observed trajectories. This resulted in **poor** performance. We then decided to use the DQN's ER buffer for generating trajectories which significantly improved the results.

This improvement in performance is due to the data in the ER buffer being essentially collected by an ensemble of networks. Therefore, there is more spread over the state-action space compared to following a constant network's policy (i.e., a DQN snapshot), resulting in a better LSTD-Q learned policy $w_{\text{S-ADP},k}$.

Evaluation: After each epoch (i.e., every million steps), we evaluated both the DQN snapshot (defined by L_k and w_k), and the DQN network where we substituted the last layer (defined by L_k and $w_{\text{S-ADP},k}$). We refer to the latter DRL network as **LSTD-Q Net**. At each epoch, we ran 50 roll-outs for the two obtained networks. A roll-out is defined as starting from the beginning of a new (random) Breakout game until the agent, controlled by the evaluated network, loses all of its lives. The policy of the agent was chosen to be ϵ-greedy on the network's outputs (estimated Q-values). Following Mnih et al. (2015), we chose $\epsilon = 0.05$.

Table 1: Experiments results

	DQN	LSTD-Q Net
Best average score	360	**386**
Std. deviation	88	**79**
Averages' p-value	3e-4	
Std. deviations' p-value	2e-5	

Results: The averaged scores for the two networks, per epoch, are presented in Figure 2. The maximal average scores and their corresponding standard deviations, and the p-values of the Wilcoxon signed-rank test for both the average scores, and the standard deviations, are given in Table 1. This non-parametric statistical test measures whether related samples differ in their means (Wilcoxon, 1945). From the presented results we remark that: **(1)** The average score of LSTD-Q Net is more stable. Once the networks reach their near-optimal performance peaks respectively (starting from around epoch 10), the LSTD-Q Net's scores plateau, whereas the DQN's scores have significant fluctuations. **(2)** LSTD-Q Net is also more stable per epoch. In general, the LSTD-Q Net's standard deviation across roll-outs, per epoch, is smaller compared to the DQN. **(3)** Finally, LSTD-Q Net achieves a higher average score than the DQN.

5 Discussion

We have presented an approach that attempts to unify DRL and S-ADP algorithms (Figure 1). Our goal is to design an RL algorithm that will (1) achieve high performance in a variety of domains, enjoy (2) safety and (3) stability as well as (4) being data efficient. We demonstrated in our experiments that re-learning the last layer of a DQN with an LSTD-Q algorithm, achieves higher performance and better stability than the DQN baseline. We note that we did not perform any policy improvement step, but only policy evaluation. This result implies that our implemetation of LSTD-Q is doing some implicit regularization by adding the diagonal term λI before solving the least squares problem.

In future work, we plan to investigate stronger S-ADP algorithms that will include policy improvement, e.g., LSPI (Lagoudakis & Parr, 2003). We also plan to explore more sophisticated unification approaches that will replace the last hidden layer of the DQN with the output of an S-ADP algorithm (Figure 1 Step (2) (**)).

Acknowledgement This research was supported in part by the European Communitys Seventh Framework Programme (FP7/2007-2013) under grant agreement 306638 (SUPREL)

References

Barto, AG and Crites, RH. Improving elevator performance using reinforcement learning. *Advances in neural information processing systems*, 8:1017–1023, 1996.

Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bertsekas, Dimitri P. Approximate dynamic programming. 2008.

Blundell, Charles, Uria, Benigno, Pritzel, Alexander, Li, Yazhe, Ruderman, Avraham, Leibo, Joel Z, Rae, Jack, Wierstra, Daan, and Hassabis, Demis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

Donahue, Jeff, Jia, Yangqing, Vinyals, Oriol, Hoffman, Judy, Zhang, Ning, Tzeng, Eric, and Darrell, Trevor. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pp. 647–655, 2013.

Lagoudakis, Michail G and Parr, Ronald. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec): 1107–1149, 2003.

Liang, Yitao, Machado, Marlos C, Talvitie, Erik, and Bowling, Michael. State of the art control of atari games using shallow reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pp. 485–493. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

Lin, Long-Ji. Reinforcement learning for robots using neural networks. 1993.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Munos, Rémi, Stepleton, Tom, Harutyunyan, Anna, and Bellemare, Marc. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1046–1054, 2016.

Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael, and Moritz, Philipp. Trust region policy optimization. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 1889–1897, 2015.

Sutton, Richard and Barto, Andrew. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Tessler, Chen, Givony, Shahar, Zahavy, Tom, Mankowitz, Daniel J, and Mannor, Shie. A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*, 2016.

Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Tsitsiklis, John N, Van Roy, Benjamin, et al. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control 42.5*, pp. 674–690, 1997.

Wang, Ziyu, de Freitas, Nando, and Lanctot, Marc. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Wilcoxon, Frank. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

Zahavy, Tom, Ben-Zrihem, Nir, and Mannor, Shie. Graying the black box: Understanding dqns. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1899–1908, 2016.

Reinforcement Learning Algorithm for Patients with Type-1 Diabetes

Phuong D. Ngo[*]
Department of Mathematics and Statistics
UiT The Artic University of Norway
Tromso, 9019, Norway
`phuong.ngo@uit.no`

Jonas Nordhaug Myhre
Department of Mathematics and Statistics
UiT The Artic University of Norway
Tromso, 9019, Norway
`jonas.n.myhre@uit.no`

Fred Godtliebsen
Department of Mathematics and Statistics
UiT The Artic University of Norway
Tromso, 9019, Norway
`fred.godtliebsen@uit.no`

Abstract

This extended abstract shows the summary of an in-silico implementation results of the reinforcement-learning optimal controller for patients with type 1 diabetes. The purpose of the proposed implementation methodology is to control the blood glucose level in the presence of meal disturbances. The controller is designed based only on interaction with the subject without knowing the description of the patient. First, data from the model is used in the no-meal scenario to learn the system and optimize its parameters. After the learning process is completed, simulations are conducted to evaluate the optimized controller when implemented on a subject model with the presence of meal disturbances. The results show that the optimal controller derived by using the reinforcement learning algorithm has significantly reduced the rise of post-meal blood glucose and maintain a desired glucose level for patients.

Keywords: Reinforcement learning, optimal adaptive control, type-1 diabetes

---

[*] Corresponding author. Email: phuong.ngo@uit.no. Tel: (+47) 92082071

1 Introduction

In type-1 diabetes, there is not enough beta-cell in the pancreas due to the destruction of the immune system, this results an increase in blood glucose. The high level glucose in blood over time leads to the damage of various organ systems. One of the major challenges in controlling the biochemical and physiologic kinetics of insulin and glucose is that the process is complex, nonlinear, and only approximately known [1]. With the development of glucose sensors and insulin pumps, real time closed loop control of the insulin-glucose system can be implemented on patients with type-1 diabetes. Hence, administering the insulin level can maintain normoglycemia. Different control algorithms have been implemented in various studies to achieve this objective. Marchetti et al. [2] used an improved proportional-integral-derivative (PID) controller for blood glucose control. Soylu [3] proposed a Mamdani type fuzzy control strategy for exogenous insulin infusion. However, most of the blood glucose controllers proposed in the literature are designed either based on a mathematical insulin-glucose model or only applied for a specific system. Since the insulin-glucose process is complex, nonlinear and varying with time, a control algorithm that does not depend on a mathematical model while be able to adapt to time varying conditions is required. Reinforcement learning has been emerged as an effective method to and control unknown nonlinear systems [4, 5].

Lewis et. al [6] has formalized methods to control unknown dynamic systems by using reinforcement learning techniques. The methods provide techniques to obtain optimal adaptive control algorithms for systems with unknown mathematical models. This paper provides an implementation of the reinforcement learning algorithms for controlling blood glucose based on an adaptive controller given in [6].

2 Mathematical Backgrounds

2.1 Reinforcement-Learning Based Optimal-Adaptive-Controller

The section summarizes the optimal adaptive control algorithm for discrete-time systems proposed in [6]. This will be used to derive an algorithm to control the blood glucose for patients with type-1 diabetes. Consider a dynamic system described by a linear discrete-time state-space equation:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{1}$$

where $\mathbf{x}_k \in R^n$ is the vector of state variables, $\mathbf{u}_k \in R^m$ is the vector of control input. A control policy and its value function are defined as:

$$\mathbf{u}_k = h(\mathbf{x}_k) \tag{2}$$

$$v^h(k) = \sum_{i=k}^{\infty} \gamma^{i-k} r(\mathbf{x}_i, \mathbf{u}_i) = \sum_{i=k}^{\infty} \left(\mathbf{x}_i^T \mathbf{Q}\mathbf{x}_i + \mathbf{u}_i^T \mathbf{R}\mathbf{u}_i\right) \tag{3}$$

where $0 < \gamma \leq 1$ is a discount factor, $\mathbf{Q} = \mathbf{Q}^T$ and $\mathbf{R} = \mathbf{R}^T$ are positive definite functions. Since the process is deterministic, the q function can be defined as follows:

$$q^h(k) = v^h(k) = r^k + \gamma v^h(k+1) \tag{4}$$

where $r^k = \mathbf{x}_k^T \mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^T \mathbf{R}\mathbf{u}_k$. Eq. 4 can be rewritten as (Bellman equation for q-function):

$$q^h(\mathbf{x}_k, \mathbf{u}_k) = r(\mathbf{x}_k, \mathbf{u}_k) + \gamma q^h(\mathbf{x}_{k+1}, \mathbf{u}_{k+1}) \tag{5}$$

The q-function can be approximated by a quadratic function of the states and inputs, or a linear function of the basis functions $\boldsymbol{\phi}(\mathbf{z})$:

$$q(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{z}) = \mathbf{w}^T(\mathbf{z} \otimes \mathbf{z}) = \mathbf{z}_k^T \mathbf{P}\mathbf{z}_k \tag{6}$$

where $\mathbf{w}$ is a vector of unknown parameters, $\boldsymbol{\phi}(\mathbf{z})$ is a vector of basis functions and $\mathbf{z}$ is the combined vector of $\mathbf{x}$ and $\mathbf{u}$:

$$\mathbf{z} = [\mathbf{x}^T \quad \mathbf{u}^T]^T \tag{7}$$

$\mathbf{P}$ is a kernel matrix with $\mathbf{P} = \mathbf{P}^T > 0$, $\mathbf{w}$ is the vector of elements of $\mathbf{P}$ and $\otimes$ is the Kronecker product. Value iteration can be used to estimate the q-function and improve the current policy [6, 7]:

$$\mathbf{w}_{j+1}^T \boldsymbol{\phi}(\mathbf{z}_k) = \left(\mathbf{x}_k^T \mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^T \mathbf{R}\mathbf{u}_k\right) + \gamma \mathbf{w}_{j+1}^T \boldsymbol{\phi}(\mathbf{z}_{k+1}) \tag{8}$$

$$h_{j+1}(\mathbf{x}_k) = \text{arg}_u\text{min}(\mathbf{w}_{j+1}^T \boldsymbol{\phi}(\mathbf{z})) \tag{9}$$

The kernel matrix $\mathbf{P}$ can be written as follows:

$$q(\mathbf{x}_k, \mathbf{u}_k) = \frac{1}{2}\begin{bmatrix}\mathbf{x}_k \\ \mathbf{u}_k\end{bmatrix}^T \mathbf{P} \begin{bmatrix}\mathbf{x}_k \\ \mathbf{u}_k\end{bmatrix} = \frac{1}{2}\begin{bmatrix}\mathbf{x}_k \\ \mathbf{u}_k\end{bmatrix}^T \begin{bmatrix}\mathbf{P}_{\mathbf{xx}} & \mathbf{P}_{\mathbf{xu}} \\ \mathbf{P}_{\mathbf{ux}} & \mathbf{P}_{\mathbf{uu}}\end{bmatrix} \begin{bmatrix}\mathbf{x}_k \\ \mathbf{u}_k\end{bmatrix} \tag{10}$$

Solving $\partial q(\mathbf{x}_k, \mathbf{u}_k)/\partial \mathbf{u}_k$ yields the action taken at time k:

$$\mathbf{u}_k = -\mathbf{P}_{\mathbf{uu}}^{-1}\mathbf{P}_{\mathbf{ux}}\mathbf{x}_k \tag{11}$$

2.2 Insulin-Glucose Kinetics Model

In this paper, both the minimal insulin-glucose kinetics model [8] and the glucose absorption model, which is part of the Hovorka model [9], are used as the simulation subject. The combined model is described as a set of differential equations:

$$\frac{dD_1(t)}{dt} = A_G D(t) - \frac{D_1(t)}{\tau_D} \tag{12}$$

$$\frac{dD_2(t)}{dt} = \frac{D_1(t)}{\tau_D} - \frac{D_2(t)}{\tau_D} \tag{13}$$

$$\frac{dg(t)}{dt} = -p_1 g(t) - x(t)[g(t) + g_b] + \frac{D_2(t)}{\tau_D} \tag{14}$$

$$\frac{dx(t)}{dt} = -p_2 x(t) + p_3 V i(t) \tag{15}$$

where the values of the parameters and constants are given in Table 1 and the variable descriptions are provided in Table 2.

Table 1: Parameters and constants of the insulin-glucose kinetics model

	Description	Value
p_1	Characteristic parameter of the minimal model	0.2 min^{-1}
p_2	Characteristic parameter of the minimal model	0.028 min^{-1}
p_3	Characteristic parameter of the minimal model	10^{-4} min^{-1}
A_G	Carbohydrate bio-availability	0.8 min^{-1}
τ_D	Glucose absorption constant	40 min
V	Plasma volume	2730 g

Table 2: Variables of the insulin-glucose kinetics model

	Description	Unit
D	Amount of CHO intake	mmol/min
D_1	Amount of glucose in compartment 1	mmol
D_2	Amount of glucose in compartment 2	mmol
$g(t)$	Deviation of glucose plasma concentration from its basal value	mmol/l
$x(t)$	Internal variable of insulin action	min^{-1}
$i(t)$	Deviation of insulin plasma concentration from the basal value	μIU/ml

The state vector of the model that will be used by the reinforcement learning algorithm is selected as follows:

$$\mathbf{x} = [g(t) \quad x(t)]^T \tag{16}$$

3 Reinforcement Learning Algorithm to Control Blood Glucose

Based on the reinforcement learning algorithm for discrete type systems, an implementation method for controlling blood glucose in patient with type-1 diabetes was derived. The algorithm includes three steps: Select an initial insulin policy, identification of q-function and control policy update.

The SIMULINK diagram for implementation of the proposed algorithm is shown in Fig. 1. In this diagram, the critic block uses the information of the state vector $\mathbf{x}$ and the action value u in oder to update the kernel matrix $\mathbf{P}$. Then, the actor block uses the kernel matrix $\mathbf{P}$ and the current state value $\mathbf{x}(k)$ to calculate the action $u(k)$. The action is interpreted as the updated insulin that will be given to the patient. The minimizing cost function is chosen as:

$$r^k = \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \tag{17}$$

The patient model also requires the meal information, which is stored in the variable scenario.meals in the workspace.

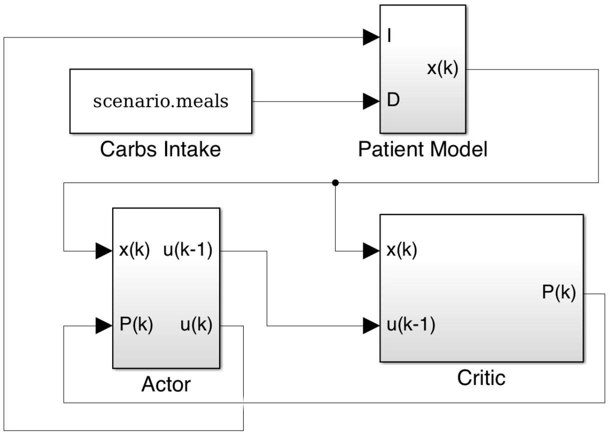


Figure 1: Reinforcement learning optimal control diagram

4 Simulation Results

4.1 Learning Stage

The learning stage was conducted to obtain the optimal insulin-glucose policy. The learning process was done using the value iteration method on the subject model without meal intake. No carbonhydrate (CHO) was simulated so that the policy is adjusted and optimized based only on the effects of insulin on glucose level. In this stage, the initial patients blood glucose is assumed to be 90 mg/dL and the target blood glucose level is 80 mg/dL.

By implementing the simulations using MATLAB and SIMULINK, an optimal controller was obtained by using reinforcement learning techniques. The results show the regulated blood glucose converges to the desired value (80 mg/dL) from the initial value (90mg/dL) in approximately 8 minutes. Based on the insulin profile proposed by the reinforcement learning algorithm, it can be seen that the proposed action has changed from the initial policy to the optimal policy during the simulation.

4.2 Evaluation Stage with Meal Disturbance

After the controller has been learned using reinforcement learning, simulations were conducted to evaluate the controller's performance when implementing on subject model under the disturbance of taking CHO meal during the control process. The total CHO taken is 50 grams in 5 mins (10 grams/min) at the beginning of the simulation. The results show that the RL-controller has significantly reduced the rising of blood glucose over the desired level of 80 mg/dL. The maximum blood glucose level in the closed loop system is approximately 90 mg/dL while the maximum blood glucose level in the open loop system is around 190 mg/dL.

5 Conclusion

The extended abstract proposes a method to implement reinforcement learning for patient with type-1 diabetes. The objective of the proposed algorithm is to control and maintain a healthy blood glucose level. A combination of the minimal and Hovorka model is used to evaluate the performance of the controller. Simulation results show that the algorithm can drive and maintain the blood glucose to a desired value in shortly after meal disturbance is introduced.

References

[1] Q. Wang, P. Molenaar, S. Harsh, K. Freeman, J. Xie, C. Gold, M. Rovine, and J. Ulbrecht, "Personalized state-space modeling of glucose dynamics for type 1 diabetes using continuously monitored glucose, insulin dose, and meal intake: an extended Kalman filter approach," *Journal of Diabetes Science and Technology*, vol. 8, pp. 331–345, mar 2014.

[2] G. Marchetti, M. Barolo, L. Jovanovic, H. Zisser, and D. E. Seborg, "An improved PID switching control strategy for type 1 diabetes," *IEEE Transactions on Biomedical Engineering*, vol. 55, no. 3, pp. 857–865, 2008.

[3] S. Soylu, K. Danisman, I. E. Sacu, and M. Alci, “Closed-loop control of blood glucose level in type-1 diabetics: A simulation study,” in *International Conference on Electrical and Electronics Engineering (ELECO)*, (Bursa), pp. 371–375, IEEE, nov 2013.

[4] M. K. Bothe, L. Dickens, K. Reichel, A. Tellmann, B. Ellger, M. Westphal, and A. A. Faisal, “The use of reinforcement learning algorithms to meet the challenges of an artificial pancreas,” *Expert Review of Medical Devices*, vol. 10, no. 5, pp. 661–673, 2013.

[5] M. De Paula, L. O. Ávila, and E. C. Martínez, “Controlling blood glucose variability under uncertainty using reinforcement learning and Gaussian processes,” *Applied Soft Computing Journal*, vol. 35, pp. 310–332, 2015.

[6] D. Vrabie, K. G. Vamvoudakis, and F. L. Lewis, *Optimal Adaptive Control and Differential Games by Reinforcement Learning Principles*. London: The Institution of Engineering and Technology, 1 ed., 2013.

[7] B. Kiumarsi, F. L. Lewis, H. Modares, and A. Karimpour, “Reinforcement Q -learning for optimal tracking control of linear discrete-time systems with unknown dynamics ,” *Automatica*, vol. 50, no. 4, pp. 1167–1175, 2014.

[8] R. N. Bergman, Y. Z. Ider, C. R. Bowden, and C. Cobelli, “Quantitative estimation of insulin sensitivity.,” *Am J Physiol Endocrinol Metab*, vol. 236, no. 6, p. E667, 1979.

[9] R. Hovorka, V. Canonico, L. J. Chassin, U. Haueter, M. Massi-Benedetti, M. Orsini Federici, T. R. Pieber, H. C. Schaller, L. Schaupp, T. Vering, M. E. Wilinska, R. Howorka, and O. M. Federici, “Nonlinear model predictive control of glucose concentration in subjects with type 1 diabetes.,” *Physiological Measurement*, vol. 25, no. 4, pp. 905–20, 2004.

Fast Adaptation of Behavior to Changing Goals with a Gamma Ensemble

Chris Reinke
Neural Computation Unit
Okinawa Institute of Science and Technology
Okinawa 904-0495, Japan
`chris.reinke@oist.jp`

Eiji Uchibe
Department of Brain Robot Interface
ATR Computational Neuroscience Laboratories
Kyoto 619-0288, Japan
`uchibe@atr.jp`

Kenji Doya
Neural Computation Unit
Okinawa Institute of Science and Technology
Okinawa 904-0495, Japan
`doya@oist.jp`

Abstract

Humans and artificial agents not only have to cope with changes in their environments, but also with changes in the goals that they want to achieve in those environments. For example, during foraging the goal could change from obtaining the most desirable food to securing food as rapidly as possible if there is time pressure. In reinforcement learning, the goal is defined by the reward function and how strongly rewards are discounted over time. If the goal changes, model-free value-based methods need to adapt their values to the new reward function or discounting strategy. This relearning is time-intensive and does not allow quick adaptation.

We propose a new model-free algorithm, the Independent Gamma-Ensemble (IGE). It is inspired by the finding that the striatum has distinct regions to encode values computed by different discount factors. Similarly, the IGE has a set of distinct modules, which are Q-functions with a different discount factors. This allows the IGE to learn and store a repertoire of different behaviors. Furthermore, it allows information about the outcome of each behavior to be decoded, making it possible to choose the best behavior for a new goal without relearning values. In a task with changing goals, the IGE outperformed a classical Q-learning agent.

The IGE is a step toward adaptive artificial agents that can cope with dynamic environments in which goals also change. Furthermore, the IGE provides a model for the potential function of the modular structure in the striatum. The striatum, which is involved in habit learning, may learn different habits in its distinct regions with different discounting factors. Depending on the context, which could be indicated by the stress level, for example, the most appropriate habit could be used without the need to relearn. This may mean that the striatum is able to learn and retain several habits for the same environment and to select them in a context-dependent manner.

Keywords: goal adaptation, model-free, Q-learning, transfer learning, value-based

1 Introduction & Problem Definition

An adaptive agent should not only be able to adapt to changes in its environment, but also to the goal that it wants to achieve in that environment. Consider, for example, one of our daily routines: going out for lunch. We have to learn the way to a good restaurant. We explore our surroundings, find several restaurants, and learn the shortest path to them. Then we decide between the different restaurants based on their reward value, e.g. the food quality, and the time to get there. We can model such tasks as episodic MDPs: (S, A, T, R) (see Fig. 1 for an example). The state space S is our position. The actions A are the directions in which we can go. We have a transition function $T(s, a, s')$ which defines what position s' we can reach from position s with action a in a fixed time, e.g. $\Delta t = 30$ seconds. If we reach a restaurant, i.e. a terminal state $G \subset S$, the reward function $R(s \in G)$ defines its food quality. We consider these factors to be external factors, which usually do not change significantly over time.

Nonetheless, there can be rapid changes in the goal that we want to achieve in this environment. One day we may want to go to the best restaurant in the area, i.e. the one that gives the highest reward. On another day, we may be experiencing stress at work and we have to judge between the restaurant quality and the time to get there. Another day we may have to entertain guests and we want to go to the closest restaurant that provides food with a certain minimum level of quality.

Current reinforcement methods include such goal formulations in the reward function R and their discounting of values over time. Model-free methods, such as Q-learning, suffer from the drawback that they need to relearn their values from new observations, i.e. experiencing the environment with the new reward function or value discounting. Learning takes time and the behavior cannot adapt from one day to the next. Model-based methods can solve this problem by maintaining a model of the external environment. The model can be used to find the best behavior for the current goal. Nonetheless, depending on the task complexity, this procedure can be computationally demanding.

We propose a new model-free algorithm, the Independent γ-Ensemble (IGE), to circumvent such problems. The algorithm is inspired by research about human intertemporal decision-making by Tanaka et al. [3]. They identified distinct brain regions in the striatum, each learning values for choices with a different discount factor. The IGE employs this concept by having several γ-modules representing Q-functions with different discount factors γ. This allows the IGE to learn and store different behaviors, e.g. going to different restaurants, and to choose the appropriate behavior, depending on the current goal.

Before describing the IGE in more detail, we introduce a distinction between external and internal reward. External reward is provided by the environment and is defined by the reward function $r_t = R(s_t)$, e.g. the food quality of a restaurant. Internal reward $\phi(r_t, t)$ depends on the goal that the agent has to achieve and modulates the external reward depending on the time t that was needed to reach it. For example, under stress we might want to reduce the external reward linearly by the time we needed to reach it: $\phi(r_t, t) = r_t - 10t$. The final objective of the agent is to maximize the sum over internal rewards. In episodic environments, the number of steps T needed to attain the maximum reward sum should also be minimized:

$$\min_T \max_\pi E_\pi \left[\sum_{t=0}^{T} \phi(r_t, t) \right] . \tag{1}$$

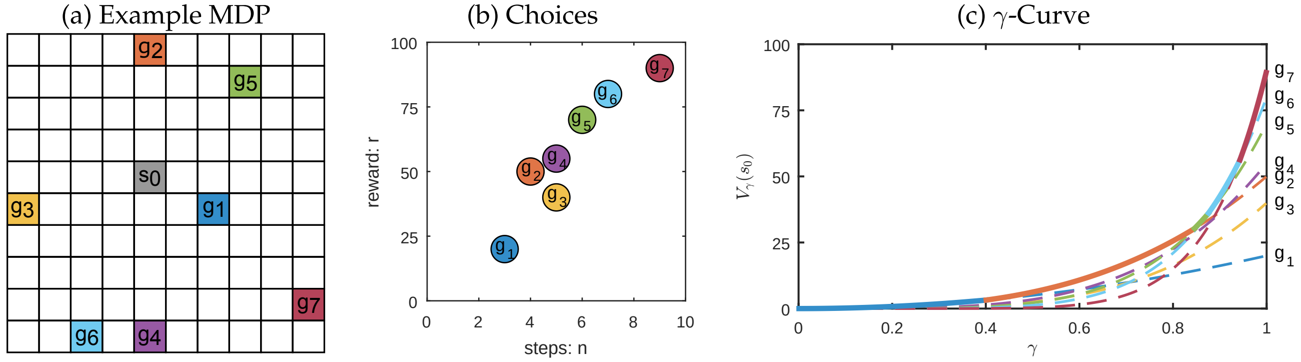


Figure 1: (a) A 2D grid world MDP with 7 terminal states $g_1, \ldots, g_7$. The agent starts in state s_0. It can move in 4 directions (north, east, south, west). Transitions result in zero reward until a terminal state is reached. (b) If the agent starts in state s_0 it has several choices to choose from. Each choice represents the minimal number of steps n and the reward r of reaching one of the possible terminal states. (c) Discounted values $V_\gamma(s_0, g) = \gamma^{n_g - 1} r_g$ for each choice for state s_0 (broken lines). The solid line represents the values the γ-Ensemble would learn ($\max_g V_\gamma(s_0, g)$). For state s_0 the ensemble will learn policies to go to terminal states g_1, g_2, g_5, g_6, g_7. It will not learn policies to go to g_3, g_4.

2 The Independent γ-Ensemble

The IGE is composed of γ-modules. Each module is a Q-function $Q_\gamma(s,a)$ with a different discount factor $\gamma \in (0,1)$. The functions are independent of each other and learn their Q-values based on the external reward $r = R(s')$. Their values are updated in parallel with the standard Q-learning rule after a transition (s,a,r,s') is observed:

$$Q_\gamma(s,a) \leftarrow Q_\gamma(s,a) + \alpha \left(r + \gamma \max_{a'} Q_\gamma(s',a') - Q_\gamma(s,a) \right). \tag{2}$$

Over time, each γ-module learns its optimal policy to maximize the discounted reward sum: $E\left[\sum_{t=0}^{T} \gamma^t r_t\right]$. Because γ differs between modules, their optimal policies differ. Modules with large γ's have a slow discounting and learn policies that result in big rewards, but that require more steps to reach. Smaller γ's have a steeper discounting and prefer smaller rewards that are reached with fewer steps. Fig. 1 (c) shows how γ-modules discount the reward for each terminal state at state s_0 for the example MDP. Modules with $\gamma < 0.4$ prefer the nearby terminal state g_1 ($r = 25$, $n_{s_0} = 3$). Whereas modules with $\gamma > 0.95$ prefer the more distant terminal state g_7 ($r = 90$, $n_{s_0} = 9$) that gives a higher reward. Therefore the IGE holds a set of policies.

In addition to providing a set of policies, the γ-ensemble can also provide information about the expected reward trajectories for those policies. This information can help to determine which policy is best for a given goal. As seen in Fig. 1 (c), modules with similar discount factors $\gamma_a \approx \gamma_b$ often result in the same trajectory, if they start from the same state s. Therefore their values $V_\gamma(s)$ are computed based on the same expected reward trajectory $R_\tau = (E[r_0], E[r_1], \ldots, E[r_T])$ with $V_\gamma(s) = E[r_0] + E[r_1]\gamma^1 + \cdots + E[r_T]\gamma^T$. Having $T+1$ modules that follow the same trajectory, their values can be defined by the matrix multiplication:

$$V = \Gamma \times R_\tau, \text{ with } V = \begin{bmatrix} V_{\gamma_0}(s) \\ \vdots \\ V_{\gamma_T}(s) \end{bmatrix}, \quad \Gamma = \begin{bmatrix} 1 & \gamma_0 & \gamma_0^2 & \cdots & \gamma_0^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \gamma_T & \gamma_T^2 & \cdots & \gamma_T^T \end{bmatrix} \text{ and } R_\tau = \begin{bmatrix} E[r_0] \\ \vdots \\ E[r_T] \end{bmatrix}.$$

Based on this formulation it is theoretically possible to decode the expected reward trajectory from the learned values with $R_\tau = \Gamma^{-1} V$. Unfortunately, this method has practical problems if applied to general MDPs. First, it is difficult to determine which γ-modules will result in the same trajectory. Furthermore, using the inverse Γ^{-1} to compute R_τ is theoretically possible, but numerically unstable because the inverse becomes ill-conditioned for large T.

Nonetheless, the idea of using several modules to decode information about their expected reward trajectory can be used in the more restricted set of deterministic, goal-only-reward MDPs. Such MDPs have deterministic state transitions $s' = T(s,a)$ and reward is only given if a terminal state is reached: $\forall s \notin G : R(s) = 0$. Although these are strict restrictions, many natural decision tasks, such as the given restaurant example, can be modeled with them. In such MDPs, the value definition is simplified to $Q_\gamma(s,a) = \gamma^{n_s} r$ with $V(s) = \max_a Q(s,a)$, where r is the expected reward gained from ending in terminal state g and n_s is the number of steps to reach it from state s. For two modules (γ_a, γ_b), which prefer the same terminal state g, their expected reward r and the number of steps n_s to reach g can be computed by solving the linear equation:

$$\begin{array}{lcccl} V_{\gamma_a}(s) = \gamma_a^{n_s} \cdot r & & & & n_s = \frac{\log(V_{\gamma_a}(s)) - \log(V_{\gamma_b}(s))}{\log(\gamma_a) - \log(\gamma_b)} \\ & \Leftrightarrow & V_{\gamma_a}(s) - \gamma_a^{n_s} = V_{\gamma_b}(s) - \gamma_b^{n_s} & \Leftrightarrow & \\ V_{\gamma_b}(s) = \gamma_b^{n_s} \cdot r & & & & r = \frac{V_{\gamma_a}(s)}{\gamma_a^{n_s}} \end{array} \tag{3}$$

Therefore, the IGE can compute $r(\gamma_i, \gamma_{i+1})$ and $n_s(\gamma_i, \gamma_{i+1})$ for the policies of neighboring γ-modules (γ_i, γ_{i+1}), because they usually have the same trajectories. A problem occurs if the trajectories are different, as for $(\gamma_a = 0.4 - \epsilon, \gamma_b = 0.4 + \epsilon)$ in Fig.1(c). In this case, the computed r and n_s are wrong. Nonetheless, these cases can be detected by comparing them to neighboring pairs. If $n_s(\gamma_a, \gamma_b) \neq n_s(\gamma_b, \gamma_c) \neq n_s(\gamma_c, \gamma_d)$, then the result of pair (γ_b, γ_c) is ignored. As a result, the IGE not only learns and stores different policies, but it also provides information about their expected rewards and the number of steps required.

This allows the IGE to adapt quickly to different task goals (Eq. 1). Given a certain goal formulation ϕ_k, it can compute the outcome, i.e. the expected internal reward sum, for each of its learned policies. Then it can select the policy that maximizes that reward sum. This is done in the beginning of an episode after the initial state s_0 is observed. First, the reward $r(\gamma_i, \gamma_{i+1})$ and the number of steps $n_{s_0}(\gamma_i, \gamma_{i+1})$ for each neighboring module pair are computed (Eq. 3). Then the internal reward sum is calculated by $\sum_{t=0}^{n_{s_0}-1} \phi_k(0,t) + \phi_k(r, n_{s_0})$. The IGE chooses then one of the modules that generates the highest internal reward sum with the fewest steps (Eq. 1). The policy of this module is then used throughout the whole episode to control the agent. If in the next episode a different goal $\phi_{j \neq k}$ is used, the IGE can immediately adapt by again choosing a module with the most appropriate policy.

3 Experiments

We tested the goal adaptation of the IGE and compared it to a time-dependent Q-learning algorithm in the MDP defined in Fig. 1. Both agents had to adapt to 8 different goals $\Phi = (\phi_1(r,n), \ldots, \phi_8(r,n))$ (Fig. 2). For convenience, we assumed that the internal reward function only gave reward when a terminal state was reached with n steps and an external reward of r. The agents had 3000 episodes to learn the first goal ϕ_1. Then the goal switched to ϕ_2 and the agents were given 1000 episodes to adapt. For each of the following goals, the agents also had 1000 episodes to adapt.

The first goal ϕ_1 is to receive the maximum external reward in an episode. The second goal ϕ_2 also maximizes external reward, but a punishment of -10 is given for each step beyond the third. ϕ_3 gives exponentially increasing punishment for more than 3 steps. The goal of ϕ_4 is to find the shortest path to the closest terminal state. For ϕ_5, the goal is the shortest path to a terminal state that gives at least an external reward of 65. Reaching a terminal state with less external reward will result in a strong negative internal reward. For ϕ_6, the goal is to find the highest external reward in less or equal 7 steps. For ϕ_7, the agent has only a limit of 5 steps. In ϕ_8, the goal is to maximize average reward.

The IGE used the procedure described in the previous section to learn different policies based on the external reward and to chose the most appropriate policy for the current goal ϕ_k. The time-dependent Q-learning learned values directly based on the internal reward. It held values for each possible time step $t = (0, \ldots, 50)$ during an episode. During learning, the values for all possible time points t were updated after an observation of (s, a, r, s') with:

$$Q(t,s,a) \leftarrow Q(t,s,a) + \alpha \left(\phi_k(r,t) + \gamma \max_{a'} Q(t+1, s', a') - Q(t,s,a) \right). \tag{4}$$

For both agents, an ϵ-Greedy action selection was used with a decaying epsilon over episodes: $\epsilon(j) = \max\left(\epsilon_0 \cdot d^j, 1\right)$.

Learning parameters of both agents were manually optimized. Both agents used the learning rate $\alpha = 1$. The IGE used 100 modules with logarithmically spaced γ parameters between 0 and 1. The parameters have a higher density toward 1 to allow better coverage of policies that result in longer paths. For the first goal ϕ_1, both agents used the same action selection parameters: $\epsilon_0 = 2, d = 0.9975$. For the IGE, action selection in all other episodes had no exploration ($\epsilon = 0$) because it could adapt immediately to a changed goal formulation. The Q-learning required exploration ($\epsilon_0 = 1, d = 0.99$) to adapt its Q-values to find the new optimal policy.

For the first goal ϕ_1, the performance of both algorithms is identical (Fig. 2), because both have to learn the Q-values for the MDP. For most of the successive tasks ($\phi_2, \ldots \phi_6, \phi_8$), the IGE can directly select the optimal policy. In contrast,

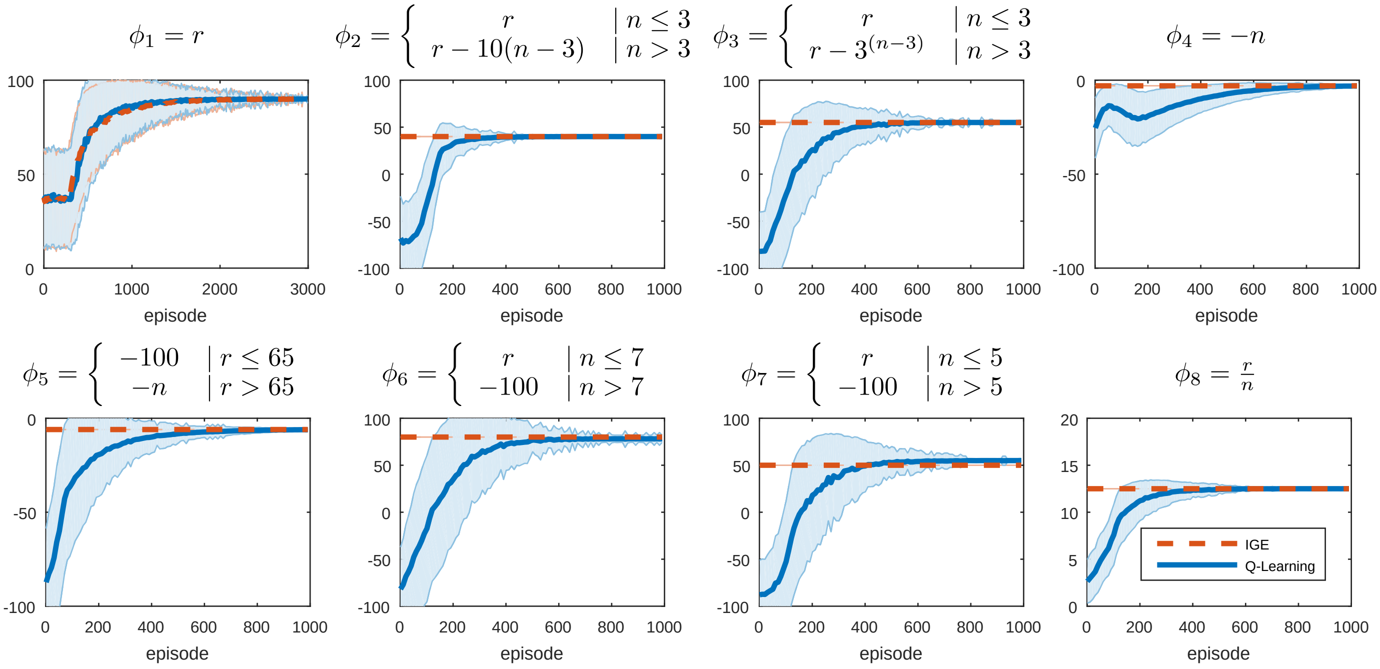


Figure 2: The learning performance of the IGE and Q-learning for all goal-formulations. The IGE can adapt directly to changes in goal-formulations ϕ, whereas Q-learning needs time to adjust its values. Performance is measured by the internal reward $\phi_k(r,n)$ that the agent achieved per episode. The plots show the mean and standard deviation of 1000 learning runs per algorithm. The minimal reward per episode was limited to -100 to make the plots more readable, because some goal formulations can result in a large negative reward during exploration.

Q-learning needed to relearn each task to solve it optimally. Task ϕ_7 is an exception. The IGE is not able to solve the task optimally, because the optimal behavior (go to g_4) is not in its policy set (Fig. 2,c).

4 Discussion

The IGE has two key features. First, it is able to learn and store different policies by using different discount factors γ for each of its modules. This allows it to switch between policies, depending on the current goal. Second, it is possible to gain information about the expected reward trajectory of the policies by using values of modules that result in the same behavior. The information about expected rewards can be used to select the most appropriate policy for a goal. The experimental results demonstrate the ability of the IGE to adapt immediately to a different goal formulation, using its stored policies and information about them. Classical model-free algorithms, such as Q-learning, need several observations to adapt to such changes.

One disadvantage of the IGE is that it can not guarantee to find the optimal policy for each goal formulation. For goal ϕ_7, the IGE fails to find the optimal policy (Fig. 2). Nonetheless, optimality can be guaranteed for a subset of goal formulations. The IGE is optimal to maximize the exponentially discounted reward sum for each of its γ factors because it uses Q-learning, which is guaranteed to converge to the optimal policy. Furthermore, it is optimal in the case of finding the shortest path or the solution with the highest external reward in episodic MDPs. Most interestingly, its convergence to the optimal policy for average reward $\frac{r}{n}$ in episodic, deterministic, goal-only-reward MDPs can be proven. For other goal formulations, the IGE can be viewed as a heuristic that does not guarantee optimality, but it often produces good results with the ability to adapt immediately to a new goal.

The IGE can be categorized as a transfer learning approach [4]. In transfer learning, a set of tasks (MDPs) M is given. The goal is to transfer knowledge from solved source tasks $M_S \subset M$ to new, unfamiliar target tasks $M_T \in M$, to improve learning performance. In our current approach, the different goal formulations Φ can be interpreted as the task set M. All tasks share the same state space, action space, and the same transition probabilities. They differ in their internal reward function ϕ_k. The IGE is able to transfer the policies learned from one task to another because the internal reward function ϕ_k between all task in M is based on a constant feature, the external reward function that all tasks in M share.

In addition to its potential use for artificial agents that need to adapt quickly to goal changes, the IGE also provides a possible model for the discount factor mapping found in the striatum by Tanaka et al. [3]. An existing model by Kurth-Nelson & Redish [2] showed that such a modular architecture can replicate hyperbolic discounting. Such discounting is often observed during intertemporal decision-making by humans. Their model is also based on modules with different discount factors, but in contrast to the IGE, those modules depend on each other and learn values for a single policy.

In contrast, the IGE proposes the idea that the striatum is composed of independent modules. The striatum, which is involved in habit learning [1], could therefore have the ability to learn and store different habitual behavior in parallel, similar to the policy set of the IGE. Furhermore, Tanaka et al. [3] showed that serotonin regulates which regions are active during a task. A low serotonin level resulted in activation of regions that have steep discounting, i.e. regions that prefer nearby rewards. Serotonin could be understood as a context signal that specifies which regions and habits are used. Because serotonin is linked to stress, it is possible that we learn different habits in parallel for different stress levels. Under low stress, we can use habits that strive for large rewards, but that require a longer time to reach; whereas under high stress we can quickly adapt by activating habits that go for short-term rewards. The advantage of such a system is that it does not need to relearn habits if its stress level changes, but to use a repertoire of habits for different stress levels.

Although the IGE is composed of independent modules, it can also replicate hyperbolic discounting as observed in the model of Kurth-Nelson & Redish [2]. This can be achieved by optimizing the average reward $\frac{r}{n}$, but instead of selecting the active module that controls the behavior in the beginning of an episode, it gets reselected after each transition.

References

[1] N. D. Daw, Y. Niv, and P. Dayan. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience*, 8(12):1704–1711, 2005.

[2] Z. Kurth-Nelson and A. D. Redish. Temporal-difference reinforcement learning with distributed representations. *PloS One*, 4(10):e7362, 2009.

[3] S. C. Tanaka, N. Schweighofer, S. Asahi, K. Shishida, Y. Okamoto, S. Yamawaki, and K. Doya. Serotonin differentially regulates short-and long-term prediction of rewards in the ventral and dorsal striatum. *PLoS One*, 2(12):e1333, 2007.

[4] M. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

Unlocking the Potential of Simulators: Design with RL in Mind

Rika Antonova **Silvia Cruciani**

KTH – Royal Institute of Technology[*]
Stockholm, Sweden
`{antonova, cruciani}@kth.se`

Abstract

Using Reinforcement Learning (RL) in simulation to construct policies useful in real life is challenging. This is often attributed to the *sequential decision making* aspect: inaccuracies in simulation accumulate over multiple steps, hence the simulated trajectories diverge from what would happen in reality.

In our work we show the need to consider another important aspect: the mismatch in simulating control. We bring attention to the need for modeling control as well as dynamics, since oversimplifying assumptions about applying actions of RL policies could make the policies fail on real-world systems.

We design a simulator for solving a pivoting task (of interest in Robotics) and demonstrate that even a simple simulator designed with RL in mind outperforms high-fidelity simulators when it comes to learning a policy that is to be deployed on a real robotic system. We show that a phenomenon that is hard to model – friction – could be exploited successfully, even when RL is performed using a simulator with a simple dynamics and noise model. Hence, we demonstrate that as long as the main sources of uncertainty are identified, it could be possible to learn policies applicable to real systems even using a simple simulator.

RL-compatible simulators could open the possibilities for applying a wide range of RL algorithms in various fields. This is important, since currently data sparsity in fields like healthcare and education frequently forces researchers and engineers to only consider sample-efficient RL approaches. Successful simulator-aided RL could increase flexibility of experimenting with RL algorithms and help applying RL policies to real-world settings in fields where data is scarce. We believe that lessons learned in Robotics could help other fields design RL-compatible simulators, so we summarize our experience and conclude with suggestions.

Keywords: Reinforcement Learning in Robotics, Learning from Simulation

Acknowledgements

We thank Christian Smith and Danica Kragic for guidance in the "Pivoting Task" work, which was supported by the European Union framework program H2020-645403 RobDREAM.

---

[*] Robotics, Perception and Learning Group: `https://www.kth.se/en/csc/forskning/rpl` at the School of Computer Science and Communication

1 Introduction

Using simulators to learn policies applicable to real-world settings is a challenge: important aspects of agent and environment dynamics might be hard and costly to model. Even if sufficient modeling knowledge and computational resources are available, inaccuracies present even in high-fidelity simulators can cause the learned policies to be useless in practice. In the case of *sequential decision making* even a small mismatch between the simulated and the real world could accumulate across multiple steps of executing a policy.

Despite these challenges, there is potential if simulators are designed with RL in mind. In our work we aim to understand what aspects of simulators are particularly important for learning RL policies. We focus first on the field of Robotics and present an approach to learn robust policies in a simulator, designed to account for uncertainty in the dynamics as well as in control of the real robot. We discuss our simulation and hardware experiments for solving a pivoting task (a robotic dexterity problem of re-orienting an object after initial grasp). Solving this task is of interest to the Robotics community; our approach to solving this task is described in detail in [1]. Here we only include the aspects of the work relevant to an interdisciplinary RLDM audience and present it as an example of RL-compatible simulator design.

We then discuss the potential of building simulators with RL in mind for other fields. In a number of fields enough domain knowledge is available to construct a task-specific or setting-specific simulator for a particular problem. It would be useful to ensure that such simulators can facilitate learning RL policies that work when transferred to real environments. This would significantly broaden the range of RL algorithms that could be used to solve a given problem – eliminating the need to restrict attention to only sample-efficient approaches (as is often done, since real-world learning is costly).

2 Why Use Simulators for Learning?

Recent success of Reinforcement Learning in games (Atari, Go) and Robotics make RL potentially the next candidate to solve challenging decision making (under uncertainty) problems in a wide variety of fields. A number of attempts to apply RL algorithms to real-world problems have been successful, but the success was mostly limited to sample-efficient approaches. It is costly and time-consuming to search for optimal policies by obtaining real-world trajectories. Batch RL – a class of algorithms that learn only from previously collected samples – could be helpful. But batch RL constitutes only a small fraction of a variety of RL algorithms. Hence, to open up the potential for using a wider range of RL algorithms we can turn to simulation.

A number of fields have already developed general-purpose simulators[1], and task-specific simulators are frequently developed for solving concrete research problems. But there is a need to facilitate successful transfer of the policies learned in simulation to real-world environments. To achieve this we can either ensure a tight enough match between simulated and real environments and use the learned policies directly, or develop data-efficient "second-stage" approaches to adjust the learned policies to the real world.

An additional consideration is current rapid advances in using deep neural networks for RL. Several recently successful algorithms, like Trust Region Policy Optimization (TRPO) [8] and Deep Deterministic Policy Gradient (DDPG) [6] are not designed to be particularly sample-efficient. Instead these allow to learn flexible control policies, and can use large neural networks as powerful policy and Q function approximators. In simulation, "deep RL" algorithms can solve problems similar in principle to those considered, for example, in Robotics. However, significant practical problems remain before they can be routinely applied to the real-world. Algorithms might not be data-efficient enough to learn on real hardware, so (pre-)training in simulation might be the only feasible solution. Success in simplified simulated settings does not immediately imply the policies could be transferred to a real-world task "as-is", but investigating the conditions necessary for successful transfer could enable approaches that succeed in real world environments.

3 Challenges and Opportunities of Simulation-based Learning: Insights from Robotics

Robotics presents a great opportunity for research in simulation-to-reality transfer for RL agents. First of all, a wide variety of general and task-specific simulators have been built to simulate various robotic systems. These simulators can model certain aspects well (e.g. the dynamics of rigid bodies with known masses and shapes). However, other aspects are harder to model precisely (e.g. dynamics of non-rigid bodies, friction). Hence, so-called "high-fidelity" simulators contain a combination of models that are precise enough to describe real-world interactions and models that are either imprecise or even misleading. This creates a challenging problem from the perspective of an uninformed RL agent. However, the simplifying aspect is that getting insights into RL algorithms is easier. Experienced roboticists can have a reasonable guess as to which aspect of the environment is hardest for the agent to learn correctly from simulation. This

---

[1] Examples of general-purpose simulators from various fields: SimStudent in education (simstudent.org); NS network simulator used for research in data networks (en.wikipedia.org/wiki/Ns_(simulator)); more than a dozen general-purpose simulators for robotics (en.wikipedia.org/wiki/Robotics_simulator).

is in contrast with other fields. For example, some leading approaches for modeling student's knowledge in the field of education have error rates not far from random guessing when applied to non-synthetic student data. So the modeling problem is hard, but it is not at all clear which aspect is at fault and what exactly is making the real student data so challenging. Another example: modeling traffic in data networks requires significant simplifications for large-scale (internet-scale) simulators. This causes difficulties if an RL agent is learning a policy for handling individual packets. By comparison, robotic simulators provide a more clear picture of what would or would not be modeled well, and there is a way to quickly test policies learned in simulation on the real environment. Such robotics experiments are frequently more accessible than running large-scale studies involving humans, where returns from the policy could be distant in time (e.g. learning outcome of a class lasting several months) or too high-stakes for experimentation (result of a treatment for a patient). Thus one strategy is to develop and test a variety of simulation-to-reality transfer approaches on robotic tasks, then use the promising approaches for costlier and longer-term studies in other fields.

We note that care should be taken to capture (even if only approximately) all the relevant environment and control dynamics in a simulator. Roboticists have looked into problems arising from under-modeling of dynamics and hardware wear-and-tear (e.g. [2]) and experimented with various approaches of injecting noise in the deterministic simulators to increase tolerance to slight modeling inaccuracies ([5] gives a summary and further references). Nonetheless, one aspect that is frequently overlooked is the application of control actions to real systems. Simple systems built in the lab can be designed to support fast and direct control. However, as the field moves to more sophisticated robotic systems that are built for a variety of tasks – oversimplifying assumptions of control could render simulation useless. Consider, for example, the Baxter robot – a medium-to-low-cost system built for small businesses and research labs. This robot is built to be relatively inexpensive, safe and versatile. Hence, in its usual operating mode, the control of the robot differs from that of high-cost industrial robots, and from that of simpler custom-built research systems. Speed and precision of controlling Baxter is traded off for lower cost of components, safe and smooth motions, versatility.

In our work, in addition to incorporating previously proposed solutions to deal with policy transfer challenges, we also emphasize the need to model delays and inaccuracies in control. We envision that as more multi-purpose medium-to-low-cost robots become widely used, it should become a standard component in building simulators. With that, we demonstrate that even a simple simulator designed with this in mind can be suitable for simulation-based RL.

4 Building RL-compatible Simulator for Solving a Pivoting Task

In this section we present a short summary of our research work described in [1], with an intent to give an example that is comprehensible to an interdisciplinary audience. We describe our approach to building a task-specific simulator for a pivoting task, then learning an RL policy in simulation and using it on the real robot[2] (without further adjustments).

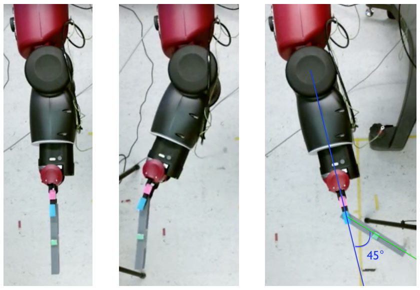


Figure 1: Pivoting task on Baxter robot.

The objective of a pivoting task is to pivot a tool to a desired angle while holding it in the gripper. This can be done by moving the arm of the robot to generate inertial forces sufficient to move the tool and, at the same time, opening or closing the gripper's fingers to change the friction at the pivoting point (gaining more precise control of the motion). The robot can use a standard planning approach to grasp the tool at a random angle ϕ_{init}. The goal is to pivot it to a target angle ϕ_{tgt}.

The limitations of the previous approaches include: only open loop control with no control of the gripping force [4], movement restricted to the vertical plane [9], gripper in a fixed position, thus motion of the tool is determined only by the gravitational torque and torsional friction [11]. All these prior approaches rely strongly on having an accurate model of the tool, as well as precise measurement and modeling of the friction. Since it is difficult to obtain these, we develop an alternative.

4.1 Dynamic and Friction Models

Our simulator system is composed of a parallel gripper attached to a link that can rotate around a single axis. This system is an under-actuated two-link planar arm, in which the under-actuated joint corresponds to the pivoting point. We assume that we can control the desired acceleration on the first joint. The dynamic model of the system is given by:

$$(I+mr^2+mlr\cos(\phi_{tl}))\ddot{\phi}_{grp} + (I+mr^2)\ddot{\phi}_{tl} + mlr\sin(\phi_{tl})\dot{\phi}_{grp}^2 + mgr\cos(\phi_{grp}+\phi_{tl}) = \tau_f, \tag{1}$$

where the variables are as follows: ϕ_{grp} and ϕ_{tl} are the angles of the first and second link respectively; $\ddot{\phi}_{grp}$ and $\ddot{\phi}_{tl}$ are their angular acceleration and $\dot{\phi}_{grp}$ is the angular velocity of the first link; l is the length of the first link; I is the tool's moment of inertia with respect to its center of mass; m is the tool's mass; r is the distance of its center of mass from the

---

[2] Video summarizing hardware experiments is available at `https://www.youtube.com/watch?v=LWSjYI9a9xw`

pivoting point; g is the gravity acceleration; τ_f is the torsional friction at the contact point between the gripper's fingers and the tool. The second link represents the tool and ϕ_{tl} is the variable we aim to control. Figure 2 illustrates this model.

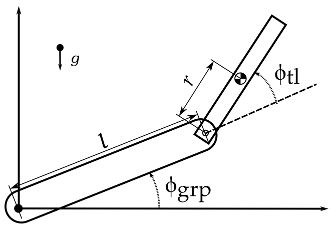


Figure 2: Model of a 2-link planar arm. First link is the gripper, second is the tool rotating around pivoting point.

Pivoting exploits friction at the contact point between the gripper and the tool to control the rotational motion. Such friction is controlled by enlarging or tightening the grasp. When the tool is not moving ($\dot{\phi}_{tl} = 0$), the static friction τ_s is modeled according to the Coulomb friction model: $|\tau_s| \leq \gamma f_n$, where γ is the coefficient of static friction and f_n is the normal force applied by gripper's fingers on the tool. When the tool moves with respect to the gripper, we model the friction torque τ_f as viscous friction and Coulomb friction [7]: $\tau_f = -\mu_v \dot{\phi}_{tl} - \mu_c f_n \operatorname{sgn}(\dot{\phi}_{tl})$, in which μ_v and μ_c are the viscous and Coulomb friction coefficients and $\operatorname{sgn}(\cdot)$ is the signum function. Since most robots are not equipped with tactile sensors to measure the normal force f_n at the contact point, as in [10] we express it as a function of the distance d_{fing} between the fingers using a linear deformation model: $f_n = k(d_0 - d_{fing})$, where k is a stiffness parameter, d_0 is the distance at which fingers initiate contact with the tool.

4.2 Learning Robust Policies

We formulate the pivoting task as an MDP. The state space is comprised of states observed at each time step t: $s_t = [\phi_{tl} - \phi_{tgt}\ ,\ \dot{\phi}_{tl}\ ,\ \phi_{grp}\ ,\ \dot{\phi}_{grp}\ ,\ d_{fing}]$, with notation as in the previous subsection. The actions are: $a_t = \{\ddot{\phi}_{grp}, d_{fing}\}$, where $\ddot{\phi}_{grp}$ is the rotational acceleration of the robot arm, and d_{fing} is the direction of the desired change in distance between the fingers of the gripper. The state transition dynamics is implemented by the simulator, but the RL algorithm does not have an explicit access to these. Reward is given at each time step t such that higher rewards are given when the angle of the tool is closer to the target angle: $r_t = \frac{-|\phi_{tl} - \phi_{tgt}|}{\phi_{RNG}} \in [-1, 1]$ (ϕ_{RNG} is a normalizing constant). A bonus reward of 1 is given when the tool is close to the target and stopped.

We aim to learn from simulated environment, while being robust to the discrepancies between the simulation and execution on the robot. For this purpose, we first built a simple custom simulator using equations from Section 4.1. To facilitate learning policies robust to uncertainty, we added 10% noise to friction values estimated for the tool modeled by the simulator. We also injected up to 10% randomized delay for arm and finger actions in simulation: used noisy linear increase in velocity (as response to acceleration action) and simulated changing fingers' distance with noisy steps.

We then trained a model-free deep RL policy search algorithm TRPO [8] on our simulated setting. TRPO has been shown to be competitive with (and sometimes outperform) other recent continuous state and action RL algorithms [3]. However, to our knowledge it has not yet been widely applied to real-world robotics tasks. While the background for the algorithm is well-motivated theoretically, the approximations made for practicality, along with challenges in achieving reasonable training results with a small-to-medium number of samples, could impair the applicability of the algorithm to learning on the robot directly. Hence we explore the approach of using TRPO for policy search in a simulated environment.

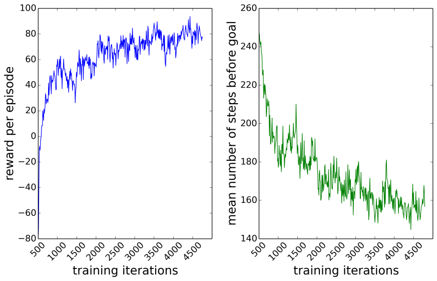
Our model of delayed and noisy control matched the behavior of real Baxter better than assuming near-instantaneous high-precision control. In contrast, learning from a high-fidelity simulator (V-REP) yielded policies that failed completely on the real robot: the tool was swayed in random directions, dropped. V-REP's oversimplification of simulating control impaired the ability of RL to learn useful policies from what was considered to be a relatively high-fidelity simulation.

4.3 Summary of Experiments

We trained TRPO (using rllab [3] implementation as a starting point and parameters as reported in [8] for simulated control tasks) with a fully connected network with 2 hidden layers (with 32, 16 nodes) for policy and Q function approximators. The motion was constrained to be linear in a horizontal plane. However, since the learning was not at all informed of the physics of the task, any other plane could be chosen and implemented by the simulator if needed.

To simulate the dynamics of the gripper arm and the tool, we used the modeling approach from Section 4.1 and injected noise during training to help learn policies robust to mismatch between the simulator and the robot. While we allowed only up to 10% noise in the variable modeling Coulomb friction, we observed that the policy was still able to retain 40% success rate of reaching the target angle when tested on settings with 250% to 500% change. This level of robustness alleviates the need to estimate friction coefficients precisely.

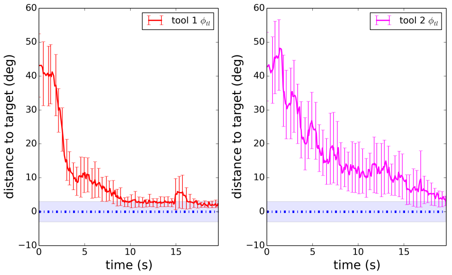
Figure 3: Evaluation of TRPO training. Initial and target angles chosen randomly from $[\frac{-\pi}{2}, \frac{\pi}{2}]$, training iterations had 50 episodes.



We deployed the learned policies on a Baxter robot and ran experiments with two objects of different materials, hence different friction properties. The friction coefficients used for modeling the tool in the simulator have been estimated only for the first tool. Hence the policy was not explicitly trained using the parameters matching those of the second tool.

We obtained a $\approx 93\%$ success rate with tool 1 and $\approx 83\%$ with tool 2. As expected, the policy performs better with the tool whose parameters were used (in a noisy manner) during training in simulation. Nonetheless, the performance using the tool not seen during training was robust as well. Fig. 4 illustrates the performance averaged over all the trials. The target is reached faster when tool 1 is used, and a bit slower when tool 2 is used. The deviation from the goal region after reaching the goal is likely due to inaccuracies in visual tracking. After reporting that the tool is in the goal region, the tracker might later report a corrected estimate, indicating further tool adjustment is needed. We observe that in such cases the policy still succeeds in further pivoting the tool to the target angle.

Figure 4: Distance to target vs time for experiments on Baxter averaged over all trials (30 trials for each tool).



5 Conclusion & Implications for other Fields for Designing RL-compatible Simulators

Our experience of building an RL-compatible simulator for a particular robotics task yields a few general suggestions.

1. *Focused modeling of the dynamics*: Since high-fidelity modeling could be expensive, one could consider modeling only the part of the dynamics relevant to the task. Domain expertise could be used to understand what is “relevant”. E.g. high-fidelity modeling of the robot arm dynamics might be unnecessary, if the task could be solved by fixing the motion to a particular plane where even simple dynamics equations are sufficient. In fields that already have general-purpose simulators such “focusing” is not always easy. The design usually does not allow to make a general-purpose simulator more task-oriented. Even if an researcher can find a restriction to the case where dynamics should be simpler – there is frequently no option in the simulator to eliminate unnecessary computations. Hence, the suggestion is to keep in mind this design consideration when building general-purpose simulators: either give the users easy control of which simulation modules to turn on/off or run automatic internal optimization of the simulator.

2. *Not oversimplifying the model of executing actions*: Neglecting to observe that the process of applying an action needs modeling could cause policies learned in even a high-fidelity simulator to fail on a real system. In contrast, including even a simple model of control application in the simulator could fix the problem. Another solution could be to reformulate an MDP/POMDP for the task to make primitive actions be something that is trivial to apply to the real system (and capture all the side-effects of applying a primitive action in the dynamics of the state transitions). This solution could work, though it might require unintuitive formulations. Regardless of the approach, the main insight is not to neglect this aspect altogether (control of real robots is not instantaneous; the action of ”give medication A” might not lead to state “took medication A” on the patient’s side; the action of “show advertisement B” might be blocked/ignored by the users).

2. *Even simple noise models help if sources of variability/uncertainty are identified*: Adding Gaussian noise to various parts of a deterministic dynamical system is an approach frequently used in robotics to “model” uncertainty. No doubt this could be too simplistic for some settings/tasks. However, since learning in simulation opens potential for cheaper experimentation – it is possible to try various ways of capturing uncertainty and identify those sufficient for learning successful policies. As we show in our experiments, a phenomenon that is hard to model – friction – could be still exploited successfully for control even with a simple dynamical model and a simple noise model. As long as the main sources of uncertainty are identified, it could be possible to learn policies applicable to real systems even from a simple simulator.

References

[1] Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic. Reinforcement learning for pivoting task. *arXiv preprint arXiv:1703.00472*, 2017.
[2] Christopher G Atkeson et al. Using local trajectory optimizers to speed up global optimization in dynamic programming. *Advances in neural information processing systems*, pages 663–663, 1994.
[3] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *arXiv preprint arXiv:1604.06778*, 2016.
[4] A. Holladay, R. Paolini, and M. T. Mason. A general framework for open-loop pivoting. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3675–3681, May 2015.
[5] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, page 0278364913495721, 2013.
[6] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
[7] H. Olsson, K. J. strm, M. Gfvert, C. Canudas De Wit, and P. Lischinsky. Friction models and friction compensation. *Eur. J. Control*, page 176, 1998.
[8] John Schulman, Sergey Levine, Philipp Moritz, Michael I Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR, abs/1502.05477*, 2015.
[9] A. Sintov and A. Shapiro. Swing-up regrasping algorithm using energy control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4888–4893, May 2016.
[10] F. E. Viña, Y. Karayiannidis, K. Pauwels, C. Smith, and D. Kragic. In-hand manipulation using gravity and controlled slip. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 5636–5641, Sept 2015.
[11] F. E. Viña, Y. Karayiannidis, C. Smith, and D. Kragic. Adaptive control for pivoting with visual and tactile feedback. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 399–406, May 2016.

Thompson Sampling for User-Guided Multi-Objective Bandits Optimization

Audrey Durand
Université Laval
audrey.durand.2@ulaval.ca

Christian Gagné
Université Laval
christian.gagne@gel.ulaval.ca

Abstract

Many real-world applications are characterized by a number of conflicting performance measures. As optimizing in a multi-objective setting leads to a set of non-dominated solutions, a preference function is required for selecting the solution with the appropriate trade-off between the objectives. This preference function is often unknown, especially when it comes from an expert human user. However, if we could provide the expert user with a *proper* estimation for each action, she would be able to pick her best choice. In this work, we tackle this problem under the user-guided multi-objective bandits formulation and we consider the Thompson sampling algorithm for providing the estimations of actions to an expert user. More specifically, we compare the extension of Thompson sampling from 1-dimensional Gaussian priors to the d-dimensional setting, for which guarantees could possibly be provided, against a fully empirical Thompson sampling without guarantees. Preliminary results highlight the potential of the latter, both given noiseless and noisy feedback. Also, since requesting information from an expert user might be costly, we tackle the problem in the context of partial feedback where the expert only provides feedback on some decisions. We study different techniques to deal with this situation. Results show Thompson sampling to be promising for the user-guided multi-objective bandits setting and that partial expert feedback is good enough and can be addressed using simple techniques. However tempting it might be to assume some given preference function, results illustrate the danger associated with a wrong assumption.

Keywords: Multi-objective optimization; Multi-armed bandits; Thompson sampling; Online decision making

Acknowledgements

This work was supported through funding from NSERC (Canada), MITACS, and E Machine Learning Inc. We also thank Julien-Charles Lévesque for insightful comments.

1 Introduction

Multi-objective optimization (MOO) [2] is a topic of great importance for real-world applications. Indeed, optimization problems are characterized by a number of conflicting, even contradictory, performance measures relevant to the task at hand. For example, when deciding on the healthcare treatment to follow for a given sick patient, a trade-off must be made between the efficiency of the treatment to heal the sickness, the side effects of the treatment, and the treatment cost. In this work, we consider problems where outcomes are stochastic and costly to evaluate, and where the performance must be optimized online. To this extent, we formulate the MOO problem as multi-objective bandits.

In this work, we tackle the MOO problem where the scalarization function is unknown and a user acts as a black box for articulating preferences. Integrating the user to the learning loop, she can provide feedback by selecting her preferred choice given a set of options – the scalarization function lying in her head. Let the *right choice* denote the option that the user would select given that she had knowledge of the Pareto-optimal set. A learning algorithm for the multi-objective bandits setting aims at learning good-enough estimations of the available options to allow the user to make the right choices. In this work, we evaluate the capability of the seminal Thompson sampling (TS) algorithm to provide such estimates. More specifically, we evaluate a fully empirical TS in the proposed setting. We also address a common issue in user-guided systems, that is the need for user assistance: asking an expert user to provide feedback on each episode may not be realistic. We therefore extend the study to the case where the expert user can only provide feedback every few episodes and evaluate different behaviors for dealing with partial feedback.

2 Multi-Objective Bandits

A multi-objective bandits problem is described by a (finite) set of actions $\mathcal{A}$, also referred to as the *design space*, each of which is associated with a d-dimensional expected outcome $\boldsymbol{\mu}_a = (\mu_{a,1}, \ldots, \mu_{a,d}) \in \mathcal{X} \in \mathbb{R}^d$. For simplicity, we assume that the *objective space* $\mathcal{X} = [0,1]^d$. In this episodic game, an agent interacts with an environment characterized by a *preference function* f. The agent iteratively chooses to perform an action a_t and obtains a noisy observation of $\boldsymbol{z}_t$.[1] Alg. 1 describes the episodic game played by an agent interacting with a user characterized by an unknown (to the agent and to the user) preference function f. If users were able to express clearly their preference function, the situation would easily be solved as a standard bandit problem. Unfortunately, this is not the case as users are generally unable to scalarize their choices and preferences. However, given several options, users can tell which one they prefer and therefore can be used as a black box to provide feedback in the learning loop.

Algorithm 1 Multi-objective bandits setting

```
On each episode $t \geq 1$:
    1. The agent selects action $a_t$ to play given $\mathcal{O}_t$.
    2. The agent observes the outcome $\boldsymbol{z}_t = \boldsymbol{\mu}_{a_t} + \boldsymbol{\xi}_t$, where $\boldsymbol{\xi}_t$ are i.i.d. random vectors.
    3. The agent updates its estimates.
    4. The agent shows options $\{\boldsymbol{\theta}_{a,t}\}_{a \in \mathcal{A}}$ to the expert user.
    5. The expert user indicates her preference $\mathcal{O}_t = \mathrm{argmax}_{a \in \mathcal{A}} f(\boldsymbol{\theta}_{a,t})$.
```

Let $\boldsymbol{x} = (x_1, \ldots, x_d)$ and $\boldsymbol{y} = (y_1, \ldots, y_d)$ denote two options. $\boldsymbol{x}$ is said to dominate, or Pareto-dominate, $\boldsymbol{y}$ (denoted $\boldsymbol{x} \succeq \boldsymbol{y}$) if and only if $x_i > y_i$ for at least one i and $x_i \geq y_i$ otherwise. Pareto-optimal options represent the best trade-offs amongst the objectives and are said to constitute the Pareto-front $\mathcal{P} = \{a : \nexists \boldsymbol{\mu}_b \succeq \boldsymbol{\mu}_a\}_{a,b \in \mathcal{A}}$. Fig. 1a shows an example of dominated and non-dominated expcted outcomes in a $d = 2$ objectives space.

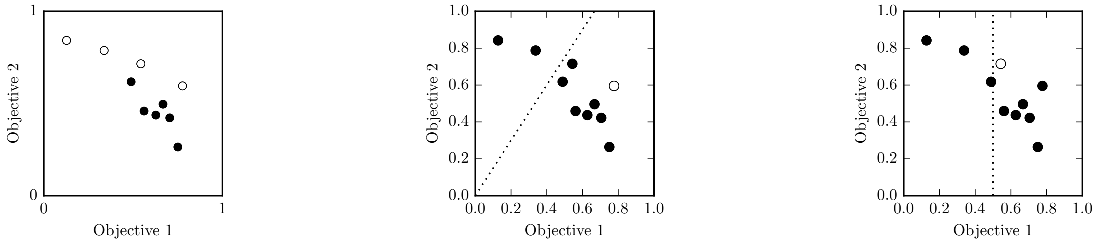
A multi-objective bandits algorithm is a (possibly randomized) method for choosing which distribution to sample from next, given a history of previous choices and obtained outcomes, $\mathcal{H}_t = \{a_s, \boldsymbol{z}_s\}_{s=1}^{t-1}$, and preference indications. Let $\mathcal{O} = \mathrm{argmax}_{a \in \mathcal{A}} f(\boldsymbol{\mu}_a) \subseteq \mathcal{P}$ and let $\star \in \mathcal{O}$ denote the optimal action, that is the right choice. The suboptimal gap $\Delta_a = f(\boldsymbol{\mu}_\star) - f(\boldsymbol{\mu}_a)$ measures the expected loss of playing action a instead of $\star$. Actions with larger Δ_a are easier to distinguish from $\star$ but they induce more regret. The goal is to minimize the cumulative regret[2]: $\mathfrak{R}(T) = \sum_{t=1}^{T}(f(\boldsymbol{\mu}_\star) - f(\boldsymbol{\mu}_{a_t}))$.

Related Works: In a first formulation of multi-objective bandits [3], the goal was to minimize a Pareto-regret metric. Also known as the ϵ-distance [5], the Pareto-regret associated with playing action a is the minimum value ϵ_a such that $\boldsymbol{\mu}_a + \epsilon_a$ is not dominated by any other actions. In other words, any actions standing on the front is considered equally good by the expert user. This is a specific instance of our more general setting where $\mathcal{O} = \mathcal{P}$. We can represent this by

---

[1] Scalars are written unbolded; vectors are boldfaced. The operators $+$, $-$, $\times$, and $\div$ applied on a vector $\boldsymbol{v} = (v_1, \ldots, v_d)$ and a scalar s correspond to the operation between each item of $\boldsymbol{v}$ and s, e.g., $\boldsymbol{v} + s = (v_1 + s, \ldots, v_d + s)$. These operators applied on two vectors $\boldsymbol{v} = (v_1, \ldots, v_d)$ and $\boldsymbol{u} = (u_1, \ldots, u_d)$ correspond to itemwise operations between $\boldsymbol{v}$ and $\boldsymbol{u}$, e.g., $\boldsymbol{v} + \boldsymbol{u} = (v_1 + u_1, \ldots, v_d + u_d)$.

[2] Also known as the scalarized regret [3].



(a) Pareto-front (b) Linear with $\alpha_1 = 0.4$, $\alpha_2 = 0.6$ (c) ϵ-constraint with $\ell = 2$, $\epsilon_1 = 0.5$

Figure 1: Pareto-front and optimal/suboptimal (white/black) means for two preference functions (dotted line).

considering the preference function $f(\boldsymbol{\mu}_\star) = 1$ and $f(\boldsymbol{\mu}_a) = 1 - \epsilon_a$, such that $\Delta_a = \epsilon_a$. With this formulation, any standard bandit algorithm that could optimize a single objective could minimize the regret regardless of the other objectives.

In MOO settings [7], the goal is to identify the Pareto-optimal set $\mathcal{P}$ without evaluating all actions. The quality of a solution $\mathcal{S}$ is typically given by the hypervolume error $V(\mathcal{P}) - V(\mathcal{S})$, where the $V(\mathcal{P})$ is the volume enclosed between the origin and $\{\boldsymbol{\mu}_a\}_{a \in \mathcal{P}}$ (and similarly for $\mathcal{S}$). However, the hypervolume error does not give information about the quality of the estimation of actions. Identifying the Pareto front alone does not guarantee that the actions are well estimated and, therefore, that an expert user choice based on these estimations would lead to the right choice.

3 Thompson Sampling

The TS [6] algorithm maintains a posterior distribution $\pi_{a,t}$ on the mean μ_a given a prior and the history $\mathcal{H}_t$. On each episode t, one option $\boldsymbol{\theta}_{a,t}$ is sampled from each posterior distribution $\pi_{a,t}$. Given that f was known, TS would choose $a_t = \operatorname{argmax}_{a \in \mathcal{A}} f(\boldsymbol{\theta}_{a,t})$. Here the options are presented to the user (step 4 of Alg. 1) and the *argmax* is provided through the expert user feedback (step 5 of Alg. 1). The algorithm chooses to sample $a_t \in \mathcal{O}_t$. Therefore $\mathbb{P}[a_t = a]$ is proportionnal to the posterior probability that a is preferred given the history $\mathcal{H}_t$.

Let $N_{a,t} = \sum_{s=1}^{t-1} \mathbb{I}[a_s = a]$ the number of times that action a has been played up to episode t. Let $\hat{\boldsymbol{\mu}}_{a,t} = \frac{\sum_{s=1}^{t-1} \boldsymbol{z}_s \mathbb{I}[a_s = a]}{N_{a,t}}$ and $\hat{\boldsymbol{\Sigma}}_{a,t} = \frac{\sum_{s=1}^{t-1} (\boldsymbol{z}_s - \hat{\boldsymbol{\mu}}_{a,t})(\boldsymbol{z}_s - \hat{\boldsymbol{\mu}}_{a,t})^\top \mathbb{I}[a_s = a]}{N_{a,t} - 1}$ denote the empirical mean and covariance, and let $\boldsymbol{\Sigma}_0$ and $\boldsymbol{\mu}_0$ denote priors. Alg. 2 shows the TS procedure from MVN priors at episode t given known covariance $\boldsymbol{\Sigma}_a$, which has theoretical guarantees for $\boldsymbol{\mu}_0 = \boldsymbol{0}_{d \times 1}$ and $\boldsymbol{\Sigma}_0 = \boldsymbol{\Sigma}_a = \boldsymbol{I}_d$ [4].[3] In this work, we consider using the empirical covariance, as depicted by Alg. 3, for some t.

Algorithm 2 TS with MVN priors and known covariance

```
for all action $a \in \mathcal{A}$ do
    compute $\tilde{\boldsymbol{\Sigma}}_{a,t} = \left(\boldsymbol{\Sigma}_0^{-1} + N_{a,t}\boldsymbol{\Sigma}_a^{-1}\right)^{-1}$
    compute $\tilde{\boldsymbol{\mu}}_{a,t} = \tilde{\boldsymbol{\Sigma}}_{a,t}\left(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0^T + N_{a,t}\boldsymbol{\Sigma}_a^{-1}\hat{\boldsymbol{\mu}}_{a,t}\right)$
    sample $\boldsymbol{\theta}_{a,t} \sim \mathcal{N}_d\left(\tilde{\boldsymbol{\mu}}_{a,t}, \tilde{\boldsymbol{\Sigma}}_{a,t}\right)$
end for
```

Algorithm 3 TS with MVN priors and empirical covariance

```
for all action $a \in \mathcal{A}$ do
    compute $\tilde{\boldsymbol{\Sigma}}_{a,t} = \left(\boldsymbol{\Sigma}_0^{-1} + N_{a,t}\hat{\boldsymbol{\Sigma}}_{a,t}^{-1}\right)^{-1}$
    compute $\tilde{\boldsymbol{\mu}}_{a,t} = \tilde{\boldsymbol{\Sigma}}_{a,t}\left(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0^T + N_{a,t}\hat{\boldsymbol{\Sigma}}_{a,t}^{-1}\hat{\boldsymbol{\mu}}_{a,t}\right)$
    sample $\boldsymbol{\theta}_{a,t} \sim \mathcal{N}_d\left(\tilde{\boldsymbol{\mu}}_{a,t}, \tilde{\boldsymbol{\Sigma}}_{a,t}\right)$
end for
```

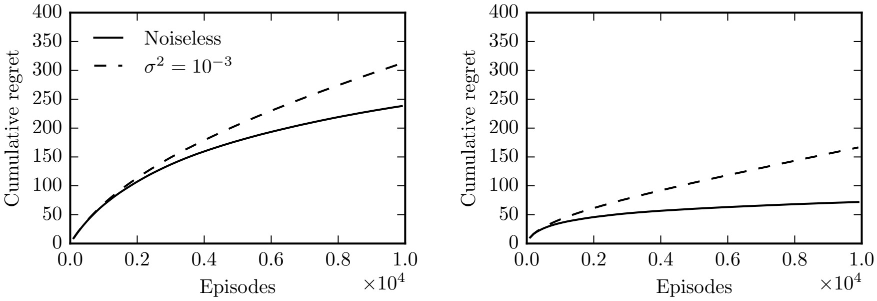
4 Empirical Evaluation

Experiments are conducted on a 2-dimensional, 10-action, setting using the linear preference function $f(\boldsymbol{x}) = 0.4x_1 + 0.6x_2$, and the ϵ-constraint function $f(\boldsymbol{x}) = x_2$ if $x_1 \geq 0.5$, $f(\boldsymbol{x}) = 0$ otherwise (Fig. 1). Note that the optimal action is different for the two preference functions. Outcomes are MVN samples of covariance $\boldsymbol{\Sigma} = \begin{bmatrix} 0.10 & 0.05 \\ 0.05 & 0.10 \end{bmatrix}$. We consider noisy feedbacks such that the preference $\tilde{O}_t = \operatorname{argmax}_{a \in \mathcal{A}} \tilde{f}(\boldsymbol{\theta}_{a,t})$, where $\tilde{f}(\boldsymbol{\theta}_{a,t}) = f(\boldsymbol{\theta}_{a,t}) + \eta_{a,t}$ with $\eta_{a,t} \sim \mathcal{N}(0, \sigma^2)$, for $\sigma^2 \in \{0, 10^{-3}\}$. All the following experiments are repeated 100 times over 10,000 episodes.

First we compare theoretical MVN TS (Alg. 2) against empirical MVN TS (Alg. 3) for recommending actions in the user-guided bandits scheme (Alg. 1). Priors $\boldsymbol{\mu}_0 = \boldsymbol{0}_{d \times 1}$ and $\boldsymbol{\Sigma}_0 = \boldsymbol{I}_d$ are used. Covariance $\boldsymbol{\Sigma}_a = \boldsymbol{I}_d$ is used for the known

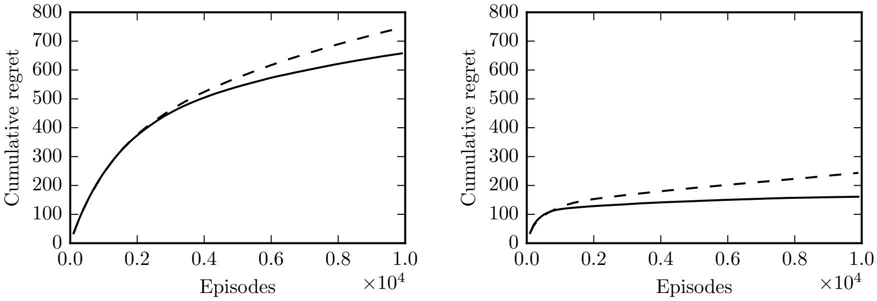
---

[3] $\boldsymbol{0}_{d \times 1}$ indicates a d-elements column vector and $\boldsymbol{I}_d$ indicates a $d \times d$ identity matrix.



(a) Theoretical TS: Linear

(b) Empirical TS: Linear



(c) Theoretical TS: ϵ-constraint

(d) Empirical TS: ϵ-constraint

Figure 2: Cumulative regret over episodes for theoretical (Alg. 2) and empirical (Alg. 3) TS with both preference functions.

covariance TS since it is not *really* known in practice[4]. Fig. 2 shows the cumulative regret of theoretical and empirical MVN TS averaged over the repetitions for both preference functions. Empirical TS appears to converge faster than theoretical TS (with a non-informative covariance), though no guarantees have been provided yet on the former.

Now we evaluate ways to deal with the situation where the expert can only provide feedback every few episodes, denoted as partial feedback. In other words, we want to *automatize* the feedback when the expert is missing. To this extent, we consider the following techniques, where τ denotes the last time feedback was provided by the expert user. In the next experiments, expert user feedback is provided every 10 episodes, that is $\tau = \min\{0, t-10\}$.

Replicate This approach considers $\mathcal{O}_t = \mathcal{O}_\tau$.

Linear assumption This approach considers the linear function $\hat{f}$ going through $\boldsymbol{\theta}_{a_\tau,\tau}$, s.t. $\mathcal{O}(t) = \operatorname{argmax}_{a \in \mathcal{A}} \hat{f}(\boldsymbol{\theta}_{a,t})$.

Squared Euclidean This approach considers $\mathcal{O}_t = \operatorname{argmin}_{a \in \mathcal{A}} \|\boldsymbol{\theta}_{a,t} - \boldsymbol{\theta}_{a_\tau,\tau}\|_2^2$.

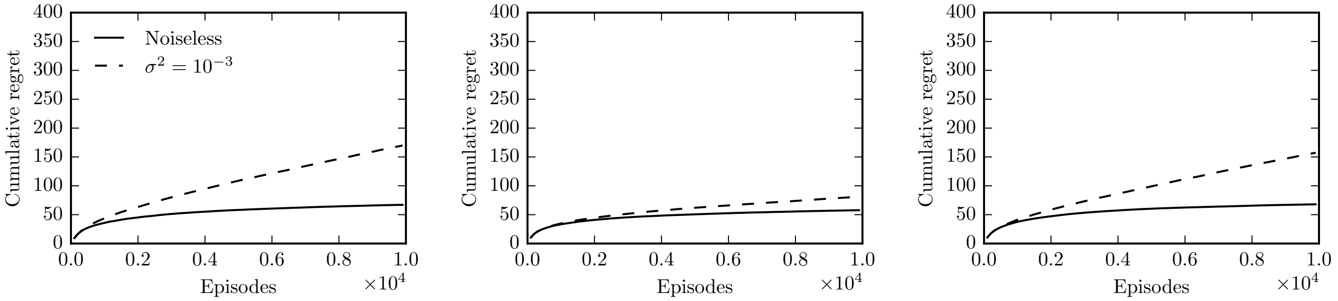
Fig. 3 shows the averaged cumulative regret of empirical MVN TS for partial expert feedback addressed using these approaches. Note the scale of Fig. 3e. We observe that the linear assumption technique performs the best when the true preference function is also linear (Fig. 3b). In fact, it even improves the performance compared with the full feedback setting. This is surprising given that the partial feedback setting uses only 10% of expert user feedback. This can be explained by the fact that the assumed linear function goes through the last selected option. Consider that $a_\tau = \star$ and $\boldsymbol{\theta}_{a_\tau,\tau} = \boldsymbol{\mu}_{a_\tau}$. With $\hat{f}$ going through $\star$ (Fig. 4a), the smallest suboptimal gap would be of 0.08 for $\hat{f}$ compared with 0.02 for f, making actions easier to distinguish and thus allowing to cumulate less regret in the partial feedback setting. On the other hand, assuming a linear preference function can lead to desastrous performance when the underlying preference function is not linear (Fig. 3e). For example, the linear function going through $\star$ in the ϵ-constraint preference setting is not maximized by $\star$ anymore (Fig. 4b vs Fig. 1c). This explains the convergence of TS toward a suboptimal action, hence the linear regret. Finally, both the replicate and squared Euclidean techniques seem to perform well. Additionnal experiments would be required to compare those in broader conditions.

5 Conclusion

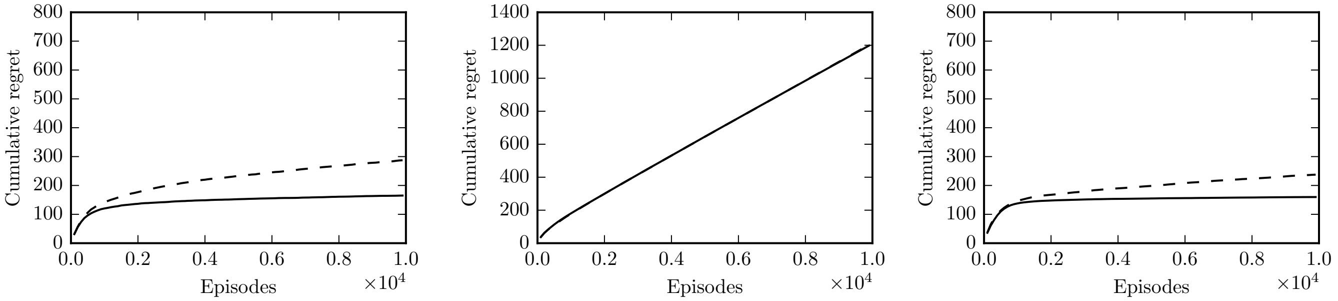
In this work we introduce the user-guided multi-objective bandits formulation. Preliminary results show that TS performs well on this problem, as expected under the assumption that user feedback acts as an *argmax* on the unknown preference function. They also highlight the potential of empirical TS, for which theoretical guarantees would be relevant. We also observe that partial expert feedback is good enough and can be dealt with using simple techniques. That

---

[4] Theoretical MVN TS therefore corresponds to a direct multi-objective extension of TS with Gaussian priors in one dimension [1].

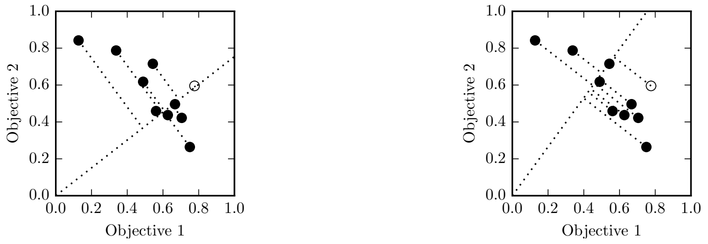


(a) Linear + Replicate (b) Linear + Linear assumption (c) Linear + Squared Euclidean



(d) ϵ-constraint + Replicate (e) ϵ-constraint + Linear assumption (f) ϵ-constraint + Squared Euclidean

Figure 3: Cumulative regret over episodes with empirical TS (Alg. 3) and partial expert feedback.



(a) $\hat{f}(\boldsymbol{x}) = 0.57x_1 + 0.43x_2$ (b) $\hat{f}(\boldsymbol{x}) = 0.43x_1 + 0.57x_2$

Figure 4: Optimal (white) and suboptimal (black) actions given the preference function $\hat{f}$.

is interesting given that a common issue with user-guided approaches is the risk of annoying the user with too many requests. However, results also show the possible danger of putting wrong assumptions on the preference function. Future work include the application of TS with MVN priors to a real-world multi-objective problem. A deeper evaluation of approaches for partial user feedback settings would also be of interest.

References

[1] S. Agrawal and N. Goyal. Further optimal regret bounds for Thompson Sampling. In *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 99–107, 2013.

[2] C. A. C. Coello, G. B. Lamont, and Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. 2007.

[3] M. M. Drugan and A. Nowe. Designing multi-objective multi-armed bandits algorithms: A study. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2013.

[4] A. Durand and C. Gagné. Estimating Quality in Multi-Objective Bandits Optimization. *ArXiv e-prints*, 2017. URL `http://arxiv.org/abs/1701.01095`.

[5] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation*, 10(3):263–82, 2002.

[6] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika Trust*, 25(3):285–294, 1933.

[7] M. Zuluaga, G. Sergent, A. Krause, and M. Püschel. Active learning for multi-objective optimization. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 462–470, 2013.

Spatial Sampling Strategies with Multiple Scientific Frames of Reference

Paul B. Reverdy
Department of Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104
`preverdy@seas.upenn.edu`

Thomas F. Shipley
Department of Psychology
Temple University
Philadelphia, PA 19122
`tshipley@temple.edu`

Daniel E. Koditschek
Department of Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, PA 19104
`kod@seas.upenn.edu`

Abstract

We study the spatial sampling strategies employed by field scientists studying aeolian processes, which are geophysical interactions between wind and terrain. As in geophysical field science in general, observations of aeolian processes are made and data gathered by carrying instruments to various locations and then deciding when and where to record a measurement. We focus on this decision-making process. Because sampling is physically laborious and time consuming, scientists often develop sampling plans in advance of deployment, i.e., employ an offline decision-making process. However, because of the unpredictable nature of field conditions, sampling strategies generally have to be updated online. By studying data from a large field deployment, we show that the offline strategies often consist of sampling along linear segments of physical space, called *transects*. We proceed by studying the sampling pattern on individual transects. For a given transect, we formulate model-based hypotheses that the scientists may be testing and derive sampling strategies that result in optimal hypothesis tests. Different underlying models lead to qualitatively different optimal sampling behavior. There is a clear mismatch between our first optimal sampling strategy and observed behavior, leading us to conjecture about other, more sophisticated hypothesis tests that may be driving expert decision-making behavior.

Keywords: Spatial sampling, frames of reference, representation, scientific decision making

Acknowledgements

This work was supported in part by NSF INSPIRE Award # 1514882.

1 Introduction

Aeolian processes, which couple the geophysical interactions between wind and terrain, are driving forces behind the production of atmospheric dust, erosion of soils, and the evolution of sand dunes in deserts [6]. These phenomena are particularly important in agriculture and climate science. Many environmental factors are relevant (wind turbulence, topography, vegetation, etc.), which leads to complex physics that are difficult to reproduce in the laboratory. Therefore, studying aeolian processes necessitates field work where scientists make observations and take measurements. There is growing reason to expect that advances in robotics [4] will soon yield a class of small legged machines capable of carrying the instruments required for these purposes of aeolian science [8]. This has prompted us to explore the prospects for developing formal models of the aeolian data collection enterprise such as might better enable human scientists to usefully direct the activities of such robotic field assistants.

A deep understanding of aeolian processes in the field remains at a relatively early stage of development; scientists have imprecise priors about what to expect when planning a field campaign. In this situation the ideal measurements would cover an area of interest at a density that would allow observation of relevant spatial variation at any scale. In the absence of infinite resources such fine-scale uniform spatial coverage of their area of interest is not practical [1]. Limited field time and financial resources will determine coverage, so a key decision-making process at the heart of field work is deciding how to allocate a limited sampling budget.

In this paper we consider sampling location data from an ongoing large-scale field campaign in Oceano, California [7]. The Oceano Dunes state park, located on the coast between Santa Barbara and San Luis Obispo, is open to the public for riding all-terrain vehicles (ATVs), which produce atmospheric dust. The field campaign was sponsored by the local air quality board in response to complaints about how ATV-generated dust was affecting local air quality. Scientists from the Desert Research Institute (DRI) were commissioned to study the processes by which dust was being generated and possible policies designed to mitigate the dust production [7, Appendix B].

The area of interest covered approximately 15 km^2, and the initial August 2013 field campaign gathered 360 samples (969 samples total over the six field campaigns in the data set). If the scientists had aimed for uniform coverage on a grid in August 2013 they would have resulted in an inter-sample distance of approximately 200 meters. However, this level of resolution is insufficient for identifying the role of the geomorphology, as it permits less than one sample per dune, whose spatial extent is on the order of 200-300 meters. Furthermore, difficulties in traversing the terrain mean that the identical amount of field resources dedicated to a uniform sampling strategy would likely have achieved fewer than 360 samples with a corresponding further decrease in resolution. Therefore, the decision makers (DMs) decided to focus on two types of individual transects, which they could sample at a higher spatial frequency. One type was oriented parallel to the prevailing wind direction (120°), and the other type was oriented parallel to the shore and thus also to the dune structures (0°). These two frames were not orthogonal due to the non-orthogonality of the two relevant orientations defined by the two physical drivers: wind direction and shore topography.

2 Field data

Figure 1 shows sampling locations from the six field campaigns conducted as part of the DRI Oceano Dunes study between 2013 and 2016. The data clearly show many linear patterns of sampling locations, which the field scientists refer to as linear *transects*. Sampling resources were focused in transects which raises three questions: 1) why were these orientations selected; 2) why were the transects located where they were; and 3) why was a regular sampling pattern used on the transects? We conducted interviews with Vicken Etyemezian, one of the senior scientists in charge of the study, to understand the decision-making process associated with these three questions. The results of the interviews are as follows.

2.1 Orientation

The initial sampling study in August 2013 was designed to measure baseline information about the rates of dust production in different areas in the park. The scientists hypothesized that there would be spatial gradients in the rates of dust production as measured by the PI-SWERL device [2]. As aeolian processes are driven by the wind, they further hypothesized that the spatial gradients would be strongly oriented along the prevailing wind direction. Therefore, the East-West transects are oriented along the prevailing wind direction and sampled at an approximately uniform 100 meter spatial resolution, selected as a compromise between achieving high spatial resolution and covering large amounts of terrain. The North-South spacing (approximately 300 meters) of these transects reflects an attempt to gather coarse North-South gradient information, while the transects oriented North-South are designed to more formally capture gradient information along a linearly independent direction [1].

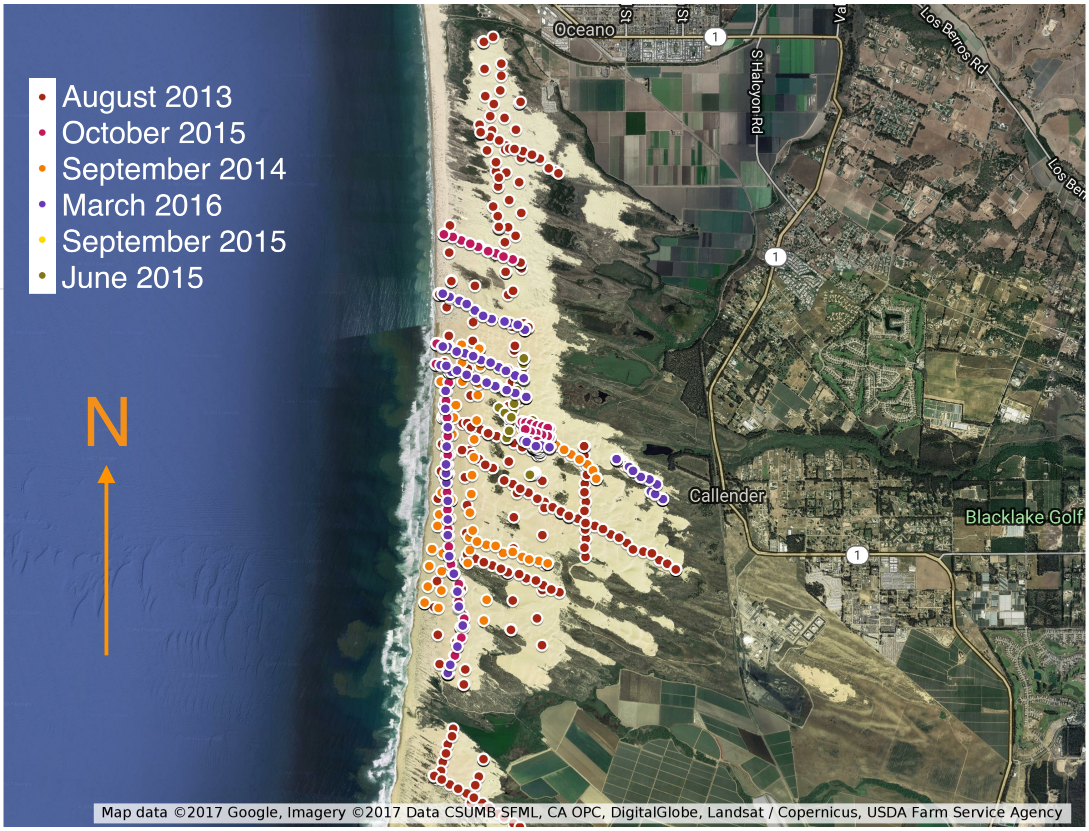
2.2 Location

The choice of location for transects is influenced by many factors, including the availability of prior meteorological measurements, land access rights, existence of vegetation, and the difficulty of traversing the physical terrain. The long 2013 transect just south of the center of the figure was designed to provide a baseline measurement of dust production upwind of the reference point for the air quality district, while the other transects were designed to provide control information in areas that were off-limits to ATV riding.

2.3 Within-transect sampling

Sampling along a given transect was done approximately every 100 meters by pre-programming GPS coordinates into a handheld receiver and then attempting to take measurements at the closest-possible location. Sometimes, as can be seen in the sampling locations at the northern edge of the park, the physical terrain made it difficult to take measurements in the desired locations.

The spatial frequency of measurement was closely related to the spatial frequency (approximately 200-300 meters) of the dune field; the resolution was chosen to ensure that there would be several measurements on each dune. In this way, the scientists hoped to capture spatial gradients that were associated with the dune topography. During a given field campaign, the sampling strategy was generally held fixed (i.e., the sampling locations were determined before going into the field). Significant changes to the sampling strategy were only made between field campaigns.



Map data ©2017 Google, Imagery ©2017 Data CSUMB SFML, CA OPC, DigitalGlobe, Landsat / Copernicus, USDA Farm Service Agency

Figure 1: Sampling locations from the six field campaigns associated with the DRI Oceano study, broken out by field campaign date. Note the clear linear patterns of sampling locations that define transects. Many of the transects have strong East-West orientations, designed to match the heading of the prevailing winds. Other transects are strongly North-South, designed to capture gradient information along a direction “orthogonal” to the prevailing wind direction. Data courtesy of Vicken Etyemezian, Desert Research Institute.

3 Hypothesis testing on a transect

In the previous section we showed that, in this study, the field scientists focused their sampling on linear transects aligned with two reference frames defined by relevant physical processes. In this section, we consider the problem of deciding where to sample on a given transect. We postulate that the decision-making process is guided by the desire to perform statistical hypothesis testing on the resulting data and investigate the spatial sampling patterns induced by maximizing the statistical power of tests associated with two different underlying models of the observed data. By comparing the induced to the observed sampling patterns, we can begin to probe the types of hypotheses the scientists are implicitly making about the structure of their data.

We restrict ourselves to a given linear transect of length ℓ and consider the problem of deciding where to sample on that interval. Let $x \in [0, \ell]$ represent locations on the transect. We suppose that the scientist is taking measurements of some quantity whose true value at location x is given by an unknown function $f(x)$. Furthermore, we assume that the measurement device produces observations

$$y = f(x) + \varepsilon, \;\; \varepsilon \sim \mathcal{N}(0, \sigma^2) \tag{1}$$

which are the true function value corrupted by zero-mean Gaussian noise with known variance σ^2.

In interviews, Dr. Etyemezian emphasized that a major goal of the sampling project was to determine the extent to which quantities thought to contribute to dust production varied with different environmental covariates, such as distance from the ocean (i.e., along a wind-oriented transect) and policy choices, such as whether or not ATV riding was permitted in a given area. This qualitative question can be cast quantitatively in terms of several different types of statistical hypothesis tests, including change point detection and tests on the coefficients of linear and nonlinear regressions. The data produced by the PI-SWERL device is quite noisy, likely because of high local variability in the quantity it is designed to measure [1]. Therefore, we regard it as unrealistic that a field campaign would be designed to perform a change point detection, as any true change point would be difficult to detect in the presence of measurement noise.

Having rejected a simple change point detection test, we consider hypothesis testing on the parameters of a regression model. In the interest of parsimony, we restrict ourselves to linear regression models. As a first hypothesis, suppose that the decision maker assumes that the unknown function $f(x)$ is linear in x:

$$f(x) = mx + b. \tag{2}$$

When there are N measurements y_i made at locations x_i for $i \in \{1, \ldots, N\}$ fitting the function (2) is an ordinary least squares (OLS) problem

$$y = X\beta$$

where $\beta = [m, b]^T$ and $X = [x_1, \ldots, x_N; 1, \ldots, 1]^T$ is the $N \times 2$ matrix of regressors. The qualitative question of whether or not f varies with x then reduces to the quantitative question of whether or not the slope m is equal to zero.

3.1 Slope test

For the linear model (2), a natural question to ask is the following: is the slope m statistically different from zero? Such a question can be answered by conducting a hypothesis test on the OLS coefficients. We adopt a frequentist statistical framework and design a sampling strategy to maximize the statistical power of the frequentist hypothesis test on m, the slope coefficient.

Assuming that the Gaussian noise ε is independent across observations i, the Gauss-Markov theorem [3] holds and the estimated regression coefficients $\hat{\beta}$ are themselves Gaussian distributed with mean β and variance $\sigma^2(X^TX)^{-1}$:

$$\hat{\beta} \sim \mathcal{N}\left(\beta, \sigma^2(X^TX)^{-1}\right).$$

Maximizing the power of the hypothesis test is equivalent to minimizing the variance of the estimator $\hat{\beta}$. The matrix $(X^TX)^{-1}$ can be expressed as

$$(X^TX)^{-1} = \frac{1}{n}\frac{1}{\overline{x^2} - \bar{x}^2}\begin{bmatrix} \overline{x^2} & -\bar{x} \\ -\bar{x} & 1 \end{bmatrix}, \tag{3}$$

where $\bar{x} = \frac{1}{N}\sum_{i=1}^N x_i$ and $\overline{x^2} = \frac{1}{N}\sum_{i=1}^N x_i^2$. A bit of additional algebra then shows that the variance of $\hat{\beta}_2$, the estimator of the slope coefficient m, is given by $\sigma^2_{\hat{\beta}_2} = 1/\sum_i(x_i - \bar{x})^2$, which is the inverse of the moment of inertia of the points x_i.

The solution to the problem of maximizing the moment of inertia $\sum_i(x_i - \bar{x})^2$ on the interval $[0, \ell]$ is simple: pick $N/2$ points at either end of the interval. Clearly, the uniform spatial sampling observed in the field is inconsistent with this solution, so we reject the hypothesis that the scientists are performing a simple slope test and seek a more sophisticated sampling criterion.

3.2 Fourier analysis

Having rejected the simple slope test we consider a more complex hypothesis test arising from linear regression. In conversation, the scientists implied that they first wanted to eliminate the possibility that there was more complex spatial behavior before testing the hypothesis that a linear model fit had zero slope, as the dune topography suggests that there may be some spatially-periodic variation in the measured variables. OLS tools can again be helpful here in the form of Fourier analysis. This is equivalent to assuming the following model for $f(x)$:

$$f(x) = \sum_{k=1}^{M} a_k \cos\left(\frac{2\pi kx}{N}\right) + \sum_{k=1}^{M} b_k \sin\left(\frac{2\pi kx}{N}\right). \tag{4}$$

Suppose that we have N uniformly-spaced samples at $x_i = i\lambda, i \in \{0, 1, \ldots, N-1\}$, where $\lambda = \ell/(N-1)$ is the spatial frequency. The well-known Nyquist sampling condition implies that we must have $M < N/2$, which provides an upper bound on the spatial frequencies that can be captured with a given sampling strategy: effectively, with a sampling frequency of λ, the highest spatial frequency that can be captured is $\lambda/2$.

Denote by $y_i, i \in \{0, 1, \ldots, N-1\}$ the N measurements of $f(x)$ taken at $x_i = i\lambda, i \in \{0, 1, \ldots, N-1\}$. Then the OLS estimators for the amplitudes a_k and b_k are the discrete Fourier transform coefficients

$$\hat{a}_k = \frac{2}{N} \sum_{i=0}^{N-1} y_i \cos\left(\frac{2\pi ki}{N}\right), \quad \hat{b}_k = \frac{2}{N} \sum_{i=0}^{N-1} y_i \sin\left(\frac{2\pi ki}{N}\right)$$

and the covariance matrix is $C = \frac{2\sigma^2}{N} I_{2M}$, which permits hypothesis testing [5, Section 4.4].

The primary experimental design decision to be made in this context is the selection of the sampling spatial frequency λ, which determines the highest spatial frequency detectable in the model. Thus, the choice of sub-dune-length-scale spatial sampling is consistent with a desire to measure spatial processes on the order of the dune length scale. Further careful investigation would be required to verify the extent to which the spatial sampling frequency is greater than twice the dune length scale, as required by the Nyquist criterion.

4 Conclusion

We investigated the spatial sampling behavior exhibited by scientists conducting research in the field. Records of field campaigns and personal interviews show that sampling tended to be performed along linear segments called *transects* and that these transects were oriented to coincide with physical drivers of the processes under study, namely the prevailing wind direction and the shoreline. We postulated that the pattern of sampling within a given transect could reveal implicit assumptions that the scientists made about the spatial structure of the scientific phenomena.

We formulated statistical hypothesis tests that might be driving the sampling behavior and considered the spatial pattern of sampling that would maximize the statistical power of the associated hypothesis test. There was a clear qualitative difference between the observed sampling behavior and that associated with a simple slope test, so we rejected that model and considered a Fourier series model. Maximizing the statistical power of a test associated with the coefficients of the Fourier series results in a regular pattern of sampling that is qualitatively similar to the observed pattern. Ongoing work will further investigate the degree to which the sampling behavior under the Fourier series model matches the observed behavior. We will conduct interviews to discover if this quantitative representation is consistent with the scientists' implicit assumptions about spatial structure and introduce other candidate models as may be suggested by the comparisons.

References

[1] V. Etyemezian. Personal communication, February 2017.

[2] V. Etyemezian, G. Nikolich, S. Ahonen, M. Pitchford, M. Sweeney, R. Purcell, J. Gillies, and H. Kuhns. The portable in situ wind erosion laboratory (PI-SWERL): A new method to measure PM 10 windblown dust properties and potential for emissions. *Atmospheric Environment*, 41(18):3789–3796, 2007.

[3] A. S. Goldberger. *A Course in Econometrics*. Harvard Univ. Press, 1991.

[4] G. C. Haynes, J. Pusey, R. Knopf, A. M. Johnson, and D. E. Koditschek. *Laboratory on legs: an architecture for adjustable morphology with legged robots*, volume 8387, pages 83870W–14. SPIE, 2012.

[5] S. M. Kay. *Fundamentals of Statistical Signal Processing, Volume I : Estimation Theory*. Prentice Hall, 1993.

[6] N. Lancaster, A. Baas, and D. Sherman. Volume 11, Aeolian geomorphology. In J. F. Shroder, editor, *Treatise on Geomorphology*. Academic Press, San Diego, 2013.

[7] S. of California. Oceano dunes state vehicular recreation area, rule 1001 draft particulate matter reduction plan. Technical report, Department of Parks and Recreation, Oceano State Park, 2013.

[8] F. Qian, D. Jerolmack, N. Lancaster, G. Nikolich, P. Reverdy, F. Roberts, Sonia, T. Shipley, R. S. Van Pelt, T. M. Zobeck, and D. Koditschek. Ground robotic measurement of aeolian processes. *Aeolian Research*, page (under review), 2016.

Neural Network Memory Architectures for Autonomous Robot Navigation

Steven W. Chen
GRASP Laboratory
University of Pennsylvania
Philadelphia, PA 19104
chenste@seas.upenn.edu

Nikolay Atanasov
GRASP Laboratory
University of Pennsylvania
Philadelphia, PA 19104
atanasov@seas.upenn.edu

Arbaaz Khan
GRASP Laboratory
University of Pennsylvania
Philadelphia, PA 19104
arbaazk@seas.upenn.edu

Konstantinos Karydis
GRASP Laboratory
University of Pennsylvania
Philadelphia, PA 19104
kkarydis@seas.upenn.edu

Daniel D. Lee
GRASP Laboratory
University of Pennsylvania
Philadelphia, PA 19104
ddlee@seas.upenn.edu

Vijay Kumar
GRASP Laboratory
University of Pennsylvania
Philadelphia, PA 19104
kumar@seas.upenn.edu

Abstract

This paper highlights the significance of including memory structures in neural networks when the latter are used to learn perception-action loops for autonomous robot navigation. Traditional navigation approaches rely on global maps of the environment to overcome cul-de-sacs and plan feasible motions. Yet, maintaining an accurate global map may be challenging in real-world settings. A possible way to mitigate this limitation is to use learning techniques that forgo hand-engineered map representations and infer appropriate control responses directly from sensed information. An important but unexplored aspect of such approaches is the effect of memory on their performance. This work is a study of memory structures for deep-neural-network-based robot navigation, and offers novel tools to train such networks from supervision and quantify their ability to generalize to unseen scenarios. We analyze the separation and generalization abilities of feedforward, long short-term memory, and differentiable neural computer networks by evaluating the generalization ability of neural networks by estimating the Vapnik-Chervonenkis (VC) dimension of maximum-margin hyperplanes trained in the feature space learned by the networks' upstream layers. We validate that these VC-dimension measures are good predictors of actual test performance. The reported method can be applied to deep learning problems beyond robotics.

Keywords: deep learning in robotics, reactive and sensor-based planning, neural network memory, generalization ability

Acknowledgements

This work is supported in part by ARL # W911NF-08-2-0004, DARPA # HR001151626/HR0011516850, ARO # W911NF-13-1-0350, and ONR # N00014-07-1-0829.

1 Introduction

Autonomous robot navigation in real-world settings involves planning and control with limited information in dynamic and partially-known environments. Traditional approaches close perception-action feedback loops by maintaining a global map representation of the environment and employing feedback motion planning algorithms (e.g., optimization-based [1], search-based [2], or sampling-based [3]). While a global map allows navigation in environments with cul-de-sacs and complex obstacle configurations, maintaining one can be challenging due to localization drift, noisy features, environment changes and limited on-board computation [4, 5].

An alternative approach to global mapping and replanning is to determine a closed-loop policy that maps the history of sensor positions and observations to the next action at each time step. The benefit of computing an action directly from the observation history is that there is no need to maintain a map, and requires just one function evaluation as opposed to a computationally-expensive search or optimization. Due to their representational power, neural-network-based learning techniques have become increasingly of interest to the robotics community as a method to encode such perception-action policies or value functions [6]. Furthermore, traditional methods for navigating in unknown environments can be used to supervise the training process by providing action labels to the neural network.

Despite successes in several fields, current network architectures are not well-suited for long-term sequential tasks [7, 8], because they are inherently reactive. Maintaining the complete observation history is not feasible, but the spatio-temporal correlation present in observation data necessitates the use of memory structures that summarize observed features (e.g., the map plays this role in traditional navigation approaches). However, the effect of memory on neural network performance and generalization ability is not well-understood. Current methods for evaluating network structures are based on empirical performance on a held-out test set. A limitation of this approach is that it is dependent on the test set choice and is prone to over-fitting during the model-selection phase [9].

Our work makes three main contributions: (1) we investigate the relationship between memory, separability and generalization ability of neural-network perception-action policies in cul-de-sac environments; (2) we evaluate the generalization ability of neural networks by estimating the Vapnik-Chervonenkis (VC) dimension [10] of maximum-margin hyperplanes (support vector machines) trained in the feature space learned by the networks' upstream layers; and (3) we develop a new parallel training algorithm for supervised learning of perception-action policies in sequential prediction problems.

2 Background

Consider a bounded connected set $\mathcal{X}$ representing the workspace of a robot. Let $\mathcal{X}^{obs}$ and $\mathcal{X}^{goal}$, called the obstacle region and the goal region, respectively, be subsets of $\mathcal{X}$. Denote the obstacle-free portion of the workspace as $\mathcal{X}^{free} := \mathcal{X} \backslash \mathcal{X}^{obs}$. The dynamics of the robot are specified by the Probability Density Function (PDF) $p_f(\cdot \mid x_t, u_t)$ of the robot state $x_{t+1} \in \mathcal{X}$ at time $t+1$ given the previous state $x_t \in \mathcal{X}$ and control input $u_t \in \mathcal{U}$. We assume that the control input space $\mathcal{U}$ is a finite discrete set.[1] The robot perceives its environment through observations $z_t \in \mathcal{Z}$ generated from a depth sensor (e.g., lidar, depth camera), whose model is specified by the PDF $p_h(\cdot \mid \mathcal{X}^{obs}, x_t)$. The information available to the robot at time t to compute the control input u_t is $i_t := (x_{0:t}, z_{0:t}, u_{0:t-1}, \mathcal{X}^{goal}) \in \mathcal{I}$, consisting of current and previous observations $z_{0:t}$, current and previous states $x_{0:t}$ and previous control inputs $u_{0:t-1}$.

Problem. *Given an initial state* $x_0 \in \mathcal{X}^{free}$ *and a goal region* $\mathcal{X}^{goal} \subset \mathcal{X}^{free}$*, find a function* $\mu : \mathcal{I} \rightarrow \mathcal{U}$*, if one exists, such that applying the control* $u_t := \mu(i_t)$ *results in a sequence of states that satisfies* $\{x_0, x_1, \ldots, x_T\} \subset \mathcal{X}^{free}$ *and* $x_T \in \mathcal{X}^{goal}$*.*

In this problem setting, the obstacle region $\mathcal{X}^{obs}$ is a partially observable state. Instead of trying to estimate it using a mapping approach, our goal is to learn a policy $\hat{\mu}$ that maps the sequence of sensor observations $z_0, z_1, \ldots$ directly to control inputs for the robot. The partial observability requires an explicit consideration of *memory* in order to learn $\hat{\mu}$ successfully.

We consider a 2-D grid world in which $\mathcal{X} \subset \mathbb{R}^2$ and $\mathcal{U} := \{down, right, up, left\}$. To investigate the need for memory, we consider U-shape cul-de-sac maps, illustrated in Fig. 1. The traditional approach to this problem is simultaneous mapping and planning. In contrast, we consider learning a feasible policy by using the outputs of an A^* path planner [11] for supervision. Let $q_t := \left(x_t, z_t, u_{t-1}, \mathcal{X}^{goal}\right)$ be the information available at time t and decompose the information state as $i_t = q_{0:t}$. Our idea is to rely on a neural network to estimate a feasible control policy $\hat{\mu}$ that maps the current

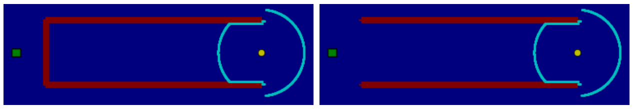


Figure 1: ***Left***: Cul-de-sac. ***Right***: Parallel Walls. The type of obstacle is unknown until the robot reaches the end of the obstacle.

---

[1] For instance, the control space $\mathcal{U}$ for a differential-drive robot in $SE(2)$ can be a set of motion primitives, parameterized by linear velocity, angular velocity and duration. For a quadrotor, $\mathcal{U}$ may be a set of short-range dynamically feasible motions.

information q_t to a control input u_t, by computing a $|\mathcal{U}|$-dimensional probability distribution over the possible controls and returning the maximum likelihood control.

We consider three neural network architectures that use different structures to represent a hidden memory state h_t in order to remember past information $q_{0:t-1}$. Deep feedforward (FF) networks, such as the multilayer perceptron [12, Ch. 6] and ConvNets, do not include a memory state h_t, and we hypothesize that the lack of a memory state will be a problem in the cul-de-sac environment because there exist two different states $i_t^{(1)}{=}q_{0:t}^{(1)}$ and $i_t^{(2)}{=}q_{0:t}^{(2)}$ (when the robot is entering and exiting the cul-de-sac) such that $\mu_t(i_t^{(1)}){\neq}\mu_t(i_t^{(2)})$ but $q_t^{(1)}{=}q_t^{(2)}$. The long-short term memory network (LSTM) [12, Ch. 10] contains a memory cell that removes and adds information to the hidden state h_t based on the previous state h_{t-1} and the current input q_t using the input, forget and update gates. The differentiable neural computer (DNC) [13] uses a more explicit representation of memory h_t in the form of a memory matrix, which may provide better memory features to separate the action classes. An external memory architecture has been shown to improve performance in natural language processing and other fields, but such an architecture has never been considered in robotics. We expect that it would be very useful for navigation where long sequences of actions may have to be backtracked.

3 Asynchronous DAgger

This section describes how we optimize the parameters θ of the networks representing the policy $\hat{\mu}(q_t, h_t; \theta)$. In sequential prediction problems, a common concern is the temporal dependence of the outputs u_t on the previous inputs $q_{0:t}$. This correlation poses a problem in stochastic gradient descent methods, as the gradient estimated from the most recent steps may no longer be representative of the true gradient. Further, the difference between the state distributions between the expert and the learner is a common concern in sequential prediction problems. A naive implementation of supervised learning will have poor performance because the states and observations encountered by the expert will be different than those encountered by the policy [14].

Inspired by the Asynchronous Advantage Actor Critic (A3C) algorithm [15], we develop an *asynchronous* variant of the DAgger algorithm [14] that breaks correlation through asynchronous gradient updates estimated from independent parallel learners. The pseudo-algorithm 1 describes the developed Asynchronous DAgger algorithm. Each parallel learner executes actions in the simulation, estimates a gradient calculated from its most recent actions, and applies that gradient asynchronously to the global neural network. Asynchronous DAgger encounters the state distribution of its current policy, as opposed to a mixture of its current and previous policies. This algorithm extends the exploitation of parallel asynchronous learners to supervised sequential prediction problems.

Algorithm 1 Asynchronous DAgger (for each learner thread)

```
// Assume global shared parameter vector $\theta$
Initialize global shared update counter $J \leftarrow 0$
Initialize thread update counter $j \leftarrow 1$
Initialize thread episode counter $t \leftarrow 1$
Initialize thread dataset $\mathcal{D} \leftarrow \emptyset$
Initialize thread network gradients $d\theta \leftarrow 0$
repeat
    repeat
        Observe $q_t$
        Execute action $u_t$ sampled from current global
            action policy $\hat{\mu}_J(q_t, h_t; \theta)$
        Retrieve optimal action $\mu(i_t)$ from expert
            and convert to standard basis vector $e_\mu$
        Add $(\hat{\mu}_J(q_t, h_t), e_\mu)$ to $\mathcal{D}$
        $j \leftarrow j + 1$
    until terminal $i_t$ or $j == j_{max}$
    for $(\hat{\mu}_J(q_t, h_t), e_\mu) \in \mathcal{D}$ do
        Accumulate gradients wrt $\theta$: $d\theta \leftarrow d\theta + \frac{d\mathcal{H}(\hat{\mu}_J, e_\mu)}{d\theta}$
            where $\mathcal{H}(\cdot, \cdot)$ is the cross-entropy loss
    if terminal $i_t$ or $t == t_{max}$ then
        Reset episode
        $t \leftarrow 1$
    Perform asynchronous update of $\theta$ using $d\theta$
    Reset thread update counter $j \leftarrow 1$
    Reset thread dataset $\mathcal{D} \leftarrow \emptyset$
    Reset thread network gradients $d\theta \leftarrow 0$
    $J \leftarrow J + 1$
until $J > J_{max}$
```

4 Generalization Ability

We present a technique for determining the generalization ability of various neural network architectures based on the maximum margin theory of generalization for support vector machines (SVM). The architecture of most deep neural networks can be broken into two parts: a hierarchy of upstream layers followed by a single readout layer (perceptron) [16]. A good neural network architecture contains upstream layers that effectively "separate" the various classes with a large margin, which the last linear readout layer can then easily classify.

Consider a binary classification problem and let $\Psi(q_i, h_i) = (\psi_1(q_i, h_i), \ldots, \psi_D(q_i, h_i))$ be a vector in feature space of dimension D, and $\boldsymbol{w} = (w_1, \ldots, w_D)$ be the vector of weights determining a hyperplane in this space. We use i instead of t to emphasize that all actions are aggregated into one data set, effectively ignoring the temporal nature of the sequential prediction. The VC-dimension η of the maximal margin hyperplanes separating N vectors $\Psi(q_1, h_1) \ldots \Psi(q_N, h_N)$ can be estimated by $\eta_{est} = R^2|\boldsymbol{w}_0|^2$, where R is the radius of the smallest sphere that contains $\Psi(q_i, h_i)$ for all i and $|\boldsymbol{w}_0|$ is the norm of the optimal weights [10]. The norm of the weights is related to the optimal margin $\Delta = \frac{1}{|\boldsymbol{w}_0|}$. Thus

Model (*radius*)	Measure	Down	Right	Up	Left
FF (*37.2*)	η_{est}	**8,558.6**	**5,740.4**	**8,016.7**	**7,698.9**
	$margin$	0.40	0.49	0.42	0.42
	nSV	340	432	438	335
	Training Err. (%)	6.7	9.0	6.7	9.0
LSTM (*4.63*)	η_{est}	**165.1**	**34.8**	**48.4**	**170.1**
	$margin$	0.36	0.79	0.67	0.36
	nSV	39	24	30	38
	Training Err. (%)	0.1	0.0	0.0	0.1
DNC LSTM (*9.42*)	η_{est}	**450.3**	**659.4**	**1221.4**	**386.1**
	$margin$	0.44	0.37	0.27	0.48
	nSV	33	28	38	26
	Training Err. (%)	0.3	0.4	0.4	0.3
***regularized* DNC LSTM (*0.94*)**	η_{est}	**14.1**	**11.2**	**11.3**	**11.9**
	$margin$	0.25	0.28	0.28	0.27
	nSV	34	17	19	33
	Training Err. (%)	0.5	0.0	0.0	0.3

Table 1: ***VC dimension estimates***: *Radius* is the radius of the smallest L_2 bounding sphere, η_{est} is the VC-dimension estimate, nSV is the number of support vectors, and *Training Err.* is calculated using the linear SVM.

Model	Map Type	*Success (%)*	*Class. Acc (%)*	A^* *Ratio*
FF	*Parallel Walls*	75.5	65.1	1.041
	Cul-de-sac	47.5	58.9	0.375
LSTM	*Parallel Walls*	100.0	76.8	1.660
	Cul-de-sac	100.0	84.1	0.891
DNC LSTM	*Parallel Walls*	56.5	65.9	1.781
	Cul-de-sac	54.5	69.7	0.892
***regularized* DNC LSTM**	***Parallel Walls***	**100.0**	**100.0**	**1.000**
	Cul-de-sac	**100.0**	**99.9**	**0.994**

Table 2: ***Empirical Test Results***: The test maps were generated with obstacle lengths of 20-120m and test for the network's abilities to extrapolate.

good generalizability (low VC-dimension) occurs when the margin Δ is large with respect to R.[2] Given a trained neural network and an evaluation data set, our approach evaluates the generalization ability by training a linear SVM in the space learned by the network's upstream layers, and using the above technique to estimate it's VC-dimension.

5 Results and Analysis

Our experiments have two purposes: (1) determine the effect of incorporating neural network memory architectures in robot navigation; and (2) evaluate the predictive capacity of our VC dimension estimates on empirical test error. Our environment is a grid-world involving cul-de-sac obstacles (see Fig. 1). The state $x_t := (\mathrm{x}_t, \mathrm{y}_t)^T$ is the 2D position of the robot. The goal is denoted by $x_t^g := (\mathrm{x}_t^g, \mathrm{y}_t^g)^T$. The robot can take four actions (down, right, up, left), and moves a distance of 1 m at each step. The robot cannot turn in place, and starts oriented toward the goal. The robot is equipped with a $360°$ field-of-view laser range finder consisting of $N_B = 144$ laser beams with maximum range 5 m that reports the distance to the closest obstacle along the beam.

We evaluate four network architectures: FF, LSTM, DNC LSTM, and regularized DNC LSTM. The inputs at each time step are the 144-dimensional LIDAR reading z_t, and the 1-dimensional position heading $atan2(\mathrm{y}_t - \mathrm{y}_t^g, \mathrm{x}_t - \mathrm{x}_t^g)$. The FF and LSTM networks have 3 layers of sizes 128 (fully-connected), 128 (fully-connected), and 128 (fully-connected or LSTM respectively). The DNC LSTM networks have an additional memory matrix and a fourth fully-connected layer of size 128. [3] The memory matrix is of size 128x32 and has 2 read heads and 1 write head. The regularized DNC LSTM has an L_2 regularization penalty of $\lambda = 0.1$ on the parameters that correspond to the external memory or LSTM layers. The last upstream layer for all 4 models has dimension 128. We use Asynchronous DAgger (Alg. 1) to train the networks. During the training phase, the obstacles had a width of 2m and length of maximum 20m, and were oriented in 0, 90, 180, and 270 °.

We compute three empirical measures of performance: (1) percentage of successful episodes; (2) percentage of correct action classification; and (3) ratio of path lengths in successful episodes compared to A^*. We also evaluate generalization ability using the method described in Section 4 following a one-vs-all binary classification framework strategy [10].

The LSTM, DNC LSTM and regularized DNC LSTM all achieve near-perfect training accuracy, but the FF does not, indicating that memory is necessary to navigate cul-de-sacs. Our VC dimension estimates (see Table 1) rank the architectures in the following order: (1) regularized DNC LSTM; (2) LSTM; (3) DNC LSTM; and (4) FF.

The test set was generated with obstacle lengths of up to 120m, or 6× the maximum training length. The test results (Table 2) confirm the predictions made by our VC dimension estimates by ranking the architectures in the same order. The regularized DNC LSTM is able to generalize almost perfectly. The LSTM is able to successfully complete all episodes, but the A^* ratio metric indicates that it is turning around before it reaches the end of the obstacle. The DNC LSTM exhibits

---

[2] We refer the reader to [10] and [17] for in-depth details and proofs. We refer the reader to [18] for details on solving for radius R.

[3] Incorporating this last fully-connected layer slightly differs from the original architecture [13], but makes the DNC a linear readout network.

similar behavior, but is only able to successfully complete $\sim 55\%$ of the episodes. These results demonstrate that the LSTM and DNC LSTM have overfit the training set, with the LSTM generalizing slightly better.

This relative degree of overfitting is reasonable, since the DNC LSTM has strictly more "memory parameters" than the LSTM. Overfitting is expected to occur as our networks are complex, yet the amount of memory to actually navigate cul-de-sac environments is relatively low. Indeed, we only need 3 states that determine when the end of the obstacle has been reached, whether the end is closed or open, and when the robot has exited the entrance of the obstacle. It is thus not surprising that regularization greatly improves the quality of the learned network.

6 Conclusion

This paper considered the problem of learning closed-loop perception-action policies for autonomous robot navigation. Unlike traditional feedback motion planning approaches that rely on accurate global maps, our approach can infer appropriate actions directly from sensed information by using a neural network policy representation. We argued that including memory in the network structure is fundamental for summarizing past information and achieving good performance as measured by its ability to separate the correct action from other choices and to generalize to unseen environments. Our main contribution is a method for evaluating neural network generalization ability through estimating the VC dimension of an SVM trained in the feature space learned by the networks' upstream layers. Finally, we proposed a new parallel training algorithm for supervised learning of closed-loop policies in sequential prediction problems. Our analysis and results demonstrated the need for and superiority of including external memory when increasing the depth of the cul-de-sacs present in the environment.

References

[1] E. Rimon and D. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.

[2] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime Dynamic A*: An Anytime, Replanning Algorithm." in *ICAPS*, 2005, pp. 262–271.

[3] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[4] S. Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2016, pp. 1484–1491.

[5] S. Thrun, "Robotic mapping: A survey," in *Exploring Artificial Intelligence in the New Millenium*, G. Lakemeyer and B. Nebel, Eds. Morgan Kaufmann, 2002.

[6] C. Richter and N. Roy, "Learning to plan for visibility in navigation of unknown environments," in *Intl. Symposium on Experimental Robotics*, 2016.

[7] S. Fusi, P. J. Drew, and L. Abbott, "Cascade models of synaptically stored memories," *Neuron*, vol. 45, no. 4, pp. 599–611, 2005.

[8] S. Ganguli, D. Huh, and H. Sompolinsky, "Memory traces in dynamical systems," *Proc. of the National Academy of Sciences*, vol. 105, no. 48, pp. 18 970–18 975, 2008.

[9] G. C. Cawley and N. L. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *Journal of Machine Learning Research*, vol. 11, no. Jul, pp. 2079–2107, 2010.

[10] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[11] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with Provable Bounds on Sub-Optimality," in *Advances in Neural Information Processing Systems (NIPS)*, 2004, pp. 767–774.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[13] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. Colmenarejo, E. Grefenstette, T. Ramalho *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.

[14] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning." in *AISTATS*, vol. 1, no. 2, 2011, p. 6.

[15] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *ArXiv e-print:1602.01783*, 2016.

[16] S. Chung, D. D. Lee, and H. Sompolinsky, "Linear readout of object manifolds," *Phys. Rev. E*, vol. 93, p. 060301, 2016.

[17] V. N. Vapnik, *Statistical Learning Theory*. Wiley-Interscience, 1998.

[18] S. Schonherr, "Quadratic programming in geometric optimization: Theory," *Implementation and Applications*, 2002.

Visualizing High-Dimensional MDPs with Model-Free Monte Carlo

Sean McGregor
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331
`RLDM@seanbmcgregor.com`

Rachel Houtman
College of Forestry
Oregon State University
Corvallis, OR 97331
`rachel.Houtman@oregonstate.edu`

Claire Montgomery
College of Forestry
Oregon State University
Corvallis, OR 97331
`claire.montgomery@oregonstate.edu`

Ronald Metoyer
College of Engineering
University of Notre Dame
Notre Dame, IN 46556
`rmetoyer@nd.edu`

Thomas G. Dietterich
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331
`tgd@oregonstate.edu`

Abstract

Policy analysts wish to visualize a range of policies for large simulator-defined Markov Decision Processes (MDPs). One visualization approach is to invoke the simulator to generate on-policy trajectories and then visualize those trajectories. When the simulator is expensive, this is not practical, and some method is required for generating trajectories for new policies without invoking the simulator. The method of Model-Free Monte Carlo (MFMC) can do this by stitching together state transitions for a new policy based on previously-sampled trajectories from other policies. This "off-policy Monte Carlo simulation" method works well when the state space has low dimension but fails as the dimension grows. This paper describes a method for factoring out some of the state and action variables so that MFMC can work in high-dimensional MDPs. The new method, MFMCi, is evaluated on a very challenging wildfire management MDP whose state space varies over more than 13 million state variables. The dimensionality of forestry domains makes MFMC unrealistic, but factorization reduces the stitching operation to 8 state features. The compact representation allows for high-fidelity visualization of policies.

Keywords: Visualization, Markov Decision Processes, Model-Free Monte Carlo,
Surrogate Modeling, State Space Factorization

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1331932.

1 Introduction

As reinforcement learning systems are increasingly deployed in the real-world, methods for justifying their ecological validity become increasingly important. For example, consider the problem of wildfire management in which land managers must decide when and where to fight fires on public lands. Our goal is to create an interactive visualization environment in which policy analysts can define various fire management polices and evaluate them through comparative visualizations. The transition dynamics of our fire management MDP are defined by a simulator that takes as input a detailed map of the landscape, an ignition location, a stream of weather conditions, and a fire fighting decision (i.e., suppress the fire vs. allow it to burn), and produces as output the resulting landscape map and associated variables (fire duration, area burned, timber value lost, fire fighting cost, etc.). The simulator also models the year-to-year growth of the trees and accumulation of fuels. Unfortunately, this simulator is extremely expensive. It can take up to 7 hours to simulate a single 100-year trajectory of fire ignitions and resulting landscapes. How can we support interactive policy analysis when the simulator is so expensive?

Our approach is to develop a surrogate model that can substitute for the simulator. We start by designing a small set of "seed policies" and invoking the slow simulator to generate several 100-year trajectories for each policy. This gives us a database of state transitions of the form (s_t, a_t, r_t, s_{t+1}), where s_t is the state at time t, a_t is the selected action, r_t is the resulting reward, and s_{t+1} is the resulting state. Given a new policy π to visualize, we apply the method of Model-Free Monte Carlo (MFMC) developed by Fonteneau et al. to simulate trajectories for π by stitching together state transitions according to a given distance metric Δ. Given a current state s and desired action $a = \pi(s)$, MFMC searches the database to find a tuple $(\tilde{s}, \tilde{a}, r, s')$ that minimizes the distance $\Delta((s, a), (\tilde{s}, \tilde{a}))$. It then uses s' as the resulting state and r as the corresponding one-step reward. We call this operation "stitching" (s, a) to $(\tilde{s}, \tilde{a})$. MFMC is guaranteed to give reasonable simulated trajectories under assumptions about the smoothness of the transition dynamics and reward function and provided that each matched tuple is removed from the database when it is used.

Fonteneau et al. (2010) apply MFMC to estimate the expected cumulative return of a new policy π by calling MFMC n times and computing the average cumulative reward of the resulting trajectories. We will refer to this as the MFMC estimate $V^{\pi}_{MFMC}(s_0)$ of $V^{\pi}(s_0)$.

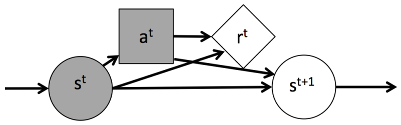
In high-dimensional spaces (i.e., where the states and actions are described by many features), MFMC breaks because of two related problems. First, distances become less informative in high-dimensional spaces. Second, the required number of seed-policy trajectories grows exponentially in the dimensionality of the space. The main technical contribution of this paper is to introduce a modified algorithm, MFMC with independencies (MFMCi), that reduces the dimensionality of the distance matching process by factoring out certain exogenous state variables and removing the features describing the action. In many applications, this can very substantially reduce the dimensionality of the matching process to the point that MFMC is again practical.

This paper is organized as follows. First, we introduce our method for factoring out exogenous variables. The method requires a modification to the way that trajectories are generated from the seed policies. Second, we conduct an experimental evaluation of MFMCi on our fire management problem. We show that MFMCi gives good performance for three different classes of policies and that for a fixed database size, it gives much more accurate visualizations.

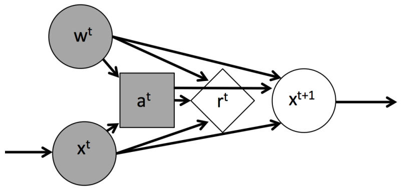
2 Factoring State to Improve MFMC

We work with the standard finite horizon undiscounted MDP (Bellman 1957; Puterman 1994), denoted by the tuple $\mathcal{M} = \langle S, A, P, R, s_0, h \rangle$. S is a finite set of states of the world; A is a finite set of possible actions that can be taken in each state; $P : S \times A \times S \mapsto [0, 1]$ is the conditional probability of entering state s' when action a is executed in state s; $R(s, a)$ is the finite reward received after performing action a in state s; s_0 is the starting state; and $\pi : S \mapsto A$ is the policy function that selects which action $a \in A$ to execute in state $s \in S$. We additionally define D as the state transition database.

In this paper, we focus on two queries about a given MDP. First, given a policy π, we wish to estimate the expected cumulative reward of executing that policy starting in state s_0: $V^{\pi}(s_0) = \mathbb{E}[\sum_{t=0}^{h} R(s_t, \pi(s_t)) | s_0, \pi]$. Second, we are interested in visualizing the distribution of the states visited at time t: $P(s_t | s_0, \pi)$. In particular, let $v^1, \ldots, v^m$ be functions that compute interesting properties of a state. For example, in our fire domain, $v^1(s)$ might compute the total area of old growth Douglas fir and $v^2(s)$ might compute the total volume of harvestable wood. Visualizing the distribution of these properties over time gives policy makers insight into how the system will evolve when it is controlled by policy π.



(a) The standard MDP transition.



(b) MDP transition with *exogenous* (w) and *Markovian* variables (x).

Figure 1: MDP probabilistic graphical models.

We now describe how we can factor the state variables of an MDP in order to reduce the dimensionality of the MFMC stitching computation. State variables can be divided into Markovian and Time-Independent random variables. A time-independent random variable x_t is exchangeable over time t and does not depend on any other random variable (including its own previous values). A (first-order) Markovian random variable x_{t+1} depends on its value x_t at the previous time step. In particular, the state variable s_{t+1} depends on s_t and the chosen action a_t. Variables can also be classified as endogenous and exogenous. The variable x_t is exogenous if its distribution is independent of $a_{t'}$ and $s_{t'} \setminus \{x'_t\}$ for all $t' \leq t$. Non-exogenous variables are endogenous. The key insight of this paper is that if a variable is time-independent and exogenous, then it can be removed from the MFMC stitching calculation as follows.

Let us factor the MDP state s into two vectors of random variables: w, which contains the time-independent, exogenous state variables and x, which contains all of the other state variables (see Figure 1). In our wildfire suppression domain, the state of the trees from one time step to another is Markovian, but our policy decisions also depend on exogenous weather events such as rain, wind, and lightning.

We can formalize this factorization as follows.

Definition 2.1. A Factored Exogenous MDP is an MDP such that the state (x, w) and next state (x', w') are related according to

$$Pr(x', w'|x, w, a) = Pr(w')Pr(x'|x, w, a). \tag{1}$$

This factorization allows us to avoid computing similarity in the complete state space s. Instead we only need to compute the similarity of the Markov state x. Without the factorization, MFMC stitches (s, a) to the $(\tilde{s}, \tilde{a})$ in the database D that minimizes a distance metric Δ, where Δ has the form $\Delta((s, a), (\tilde{s}, \tilde{a})) \mapsto \mathbb{R}^+$. Our new algorithm, MFMCi, makes its stitching decisions using only the Markov state. It stitches the current state x by finding the tuple $(\tilde{x}, \tilde{w}, a, r, x')$ that minimizes the lower-dimensional distance metric $\Delta_i(x, \tilde{x})$. MFMCi then adopts $(\tilde{x}, \tilde{w})$ as the current state, computes the policy action $\tilde{a} = \pi(\tilde{x}, \tilde{w})$, and then makes a transition to x' with reward r. The rationale for replacing x by $\tilde{x}$ is the same as in MFMC, namely that it is the nearest state from the database D. The rationale for replacing w by $\tilde{w}$ is that both w and $\tilde{w}$ are exchangeable draws from the exogenous time-independent distribution $P(w)$, so they can be swapped without changing the distribution of simulated paths.

There is one subtlety that can introduce bias into the simulated trajectories. What happens when the action $\tilde{a} = \pi(\tilde{x}, \tilde{w})$ is not equal to the action a in the database tuple $(\tilde{x}, \tilde{w}, a, r, x', w')$? One approach would be to require that $a = \tilde{a}$ and keep rejecting candidate tuples until we find one that satisfies this constraint. We call this method, "Biased MFMCi", because doing this introduces a bias. Consider again the graphical model in Figure 1. When we use the value of a to decide whether to accept $\tilde{w}$, this couples $\tilde{w}$ and $\tilde{x}$ so that they are no longer independent.

An alternative to Biased MFMCi is to change how we generate the database D to ensure that for every state $(\tilde{x}, \tilde{w})$, there is always a tuple $(\tilde{x}, \tilde{w}, a, r, x', w')$ for every possible action a. To do this, as we execute a trajectory following policy π, we simulate the result state (x', w') and reward r for each possible action a and not just the action $a = \pi(x, w)$ dictated by the policy. We call this method "Debiased MFMCi". This requires drawing more samples during database construction, but it restores the independence of $\tilde{w}$ from $\tilde{x}$.

Fonteneau et al. (2013; 2014; 2010) derived bias and variance bounds on the MFMC value estimate $V^{\pi}_{MFMC}(s_0)$. We prove improved bounds for debiased MFMCi in McGregor et al. (2017a).

3 Experimental Evaluation

In our experiments we test whether we can generate accurate trajectory visualizations for a wildfire, timber, vegetation, and weather simulator of Houtman et al. The aim of the wildfire management simulator is to help US Forest Service land managers decide whether they suppress a wildfire on National Forest lands. Each 100-year trajectory takes up to 7 hours to simulate.

The landscape is comprised of approximately one million pixels, each with 13 state variables. When a fire is ignited by lightning, the policy must choose between two actions: *Suppress* (fight the fire) and *Let Burn* (do nothing). Hence, $|A| = 2$.

The simulator spreads wildfires with the FARSITE fire model (Finney 1998) according to the surrounding pixel variables (X) and the hourly weather. MFMCi can treat the weather variables and the ignition location

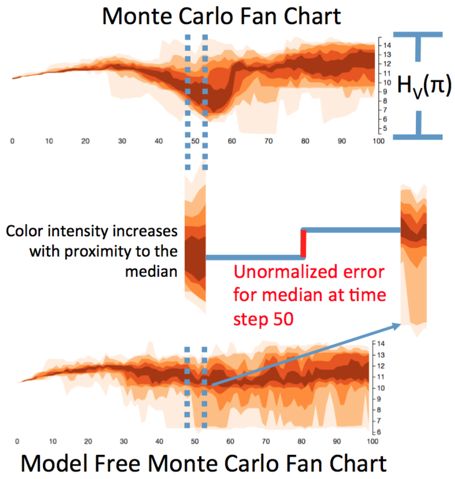


Figure 2: Top: A fan chart generated by Monte Carlo simulations from the expensive simulator. Bottom: A fan chart generated from the MFMC surrogate model. The x axis is the time step and the y axis is the value of the state variable at each time step. Each change in color shows a quantile boundary for a set of trajectories generated under policy π. Middle: Error measure is the distance between the median of the Monte Carlo simulations (left) and the median of the MFMC/MFMCi surrogate simulations (right). The error is normalized across fan charts according to $H_v(\pi)$, which is the Monte Carlo fan chart height for policy π and variable v.

as exogenous variables because the decision to fight (or not fight) a fire has no influence on weather or ignition locations. Further, changes in the Markov state do not influence the weather or the spatial probability of lightening strikes.

After computing the extent of the wildfire on the landscape, the simulator applies a cached version of the Forest Vegetation Simulator (Dixon 2002) to update the vegetation of the individual pixels. Finally, a harvest scheduler selects pixels to harvest for timber value.

We constructed three policy classes that map fires to fire suppression decisions. We label these policies INTENSITY, FUEL, and LOCATION. The INTENSITY policy suppresses fires based on the weather conditions at the time of the ignition and the number of days remaining in the fire season. The FUEL policy suppresses fires when the landscape accumulates sufficient high-fuel pixels. The LOCATION policy suppresses fires starting on the top half of the landscape, and allows fires on the bottom half of the landscape to burn (which mimics the situation that arises when houses and other buildings occupy part of the landscape). We selected these policy classes because they are functions of different components of the Markov and exogenous state.

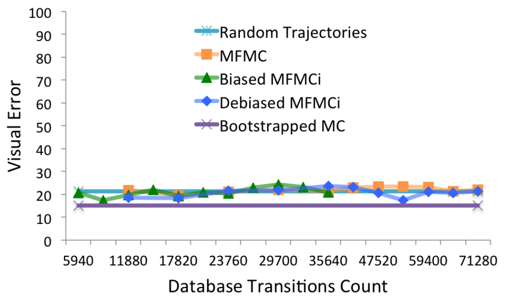
We seed the database with one trajectory for each of 360 policies whose parameters are sampled according to a grid over the INTENSITY policy space. We evaluate MFMCi by generating 30 trajectories for each policy from the ground truth simulator.

For our distance metric Δ_i, we use a weighted Euclidean distance computed over the mean/variance standardized values of 7 landscape features. An additional feature, the time step (*Year*), is added to the distance metric with a very large weight to ensure that MFMCi will only stitch from one state to another if the time steps match.

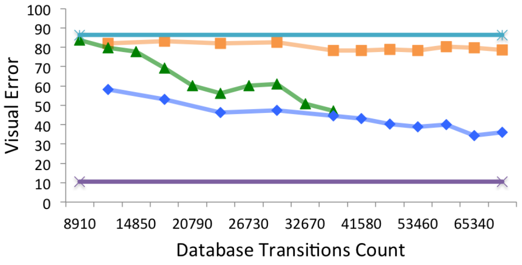
To visualize the trajectories, we employ the visualization tool MDPVIS (McGregor et al. 2015). The key visualization in MDPVIS is the fan chart, which depicts various quantiles of the set of trajectories as a function of time (see Figure 2). To evaluate the quality of the fan charts generated using surrogate trajectories, we define visual fidelity error in terms of the difference in vertical position between the true median as sampled with Monte Carlo trajectories and its position under the surrogate. Specifically, we define $\text{error}(v,t)$ as the offset between the Monte Carlo location of the median and its MFMCi-modeled location for state variable v in time step t. We normalize the error by the height of the fan chart for the rendered policy ($H_v(\pi)$). The weighted error is thus $\sum_{v\in S}\sum_{t=0}^{h}\frac{\text{error}(v,t)}{H_v(\pi)}$. This error is measured for 20 variables related to the counts of burned pixels, fire suppression expenses, timber loss, timber harvest, and landscape ecology.

3.1 Experimental Results

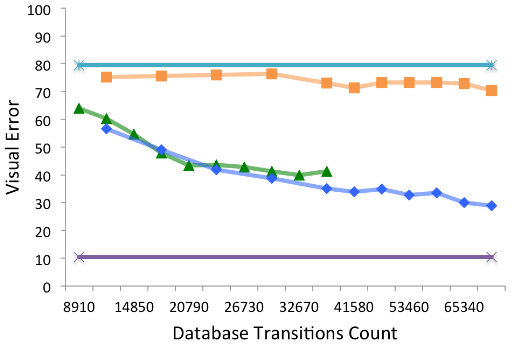
We evaluated the visual fidelity under three settings: (a) debiased MFMCi (exogenous variables excluded from the distance metric Δ_i; debiasing tuples included in the database D), (b) MFMC (exogenous variables included in Δ), and (c) biased MFMCi (exogenous variables excluded from Δ_i and the extra debiasing tuples removed from D). We also compare against two baselines that explore the upper and lower bounds of the visual error. First, we show that the lower bound on visual error is not zero. Although each policy has true quantile values at every time step, estimating these quantiles with 30 trajectories is inherently noisy. We estimate the achievable visual fidelity by bootstrap resampling the 30 ground truth trajectories and report the average visual fidelity error. Second, we check whether the error introduced by stitching is worse than visualizing a set of random database trajectories. Thus the bootstrap resample forms a lower bound on the error, and comparison to the random trajectories detects stitching failure. Figure 3 plots "learning curves" that plot the visualization error as a function of the size of the database D. The ideal learning curve should show a rapid decrease in visual fidelity error as $|D|$ grows.



(a) Visual fidelity errors for a weather *intensity* policy class. Fires are suppressed based on a combination of the weather and how much time is left in the fire season.



(b) Visual fidelity errors for a ignition *location* policy class. Fires are always suppressed if they start on the top half of the landscape, otherwise they are always allowed to burn.



(c) Visual fidelity errors for a *fuel* accumulation policy class. Fires are always suppressed if the landscape is at least 30 percent in high fuels, otherwise the fire is allowed to burn.

Figure 3: Policy classes for the wildfire domain under a variety of distance metrics and sampling procedures.

4 Discussion

For each policy class, we chose one target policy from that class and measured how well the behavior of that policy could be simulated by our MFMC variants. Recall that the database of transitions was generated using a range of INTENSITY policies. When we apply the MFMC variants to generate trajectories for an INTENSITY policy, all methods (including random trajectory sampling) produce an accurate representation of the median for MDPVIS. When the database trajectories do not match the target policy, MFMCi outperforms MFMC. For some policies, the debiased database outperforms the biased databases, but the difference decreases with additional database samples. Next we explore these findings in more depth.

INTENSITY **Policy.** Figure 3a shows the results of simulating an INTENSITY policy that suppresses all late-season moderate intensity fires ($\sim$60 percent of fires). There are many trajectories in the database that agree with the target policy on the majority of fires. Thus, to simulate the target policy it is sufficient to find a policy with a high level of agreement and then sample the entire trajectory. This is exactly what MFMC, MFMCi, and Biased MFMCi do. All of them stitch to a good matching trajectory and then follow it, so they all give accurate visualizations as indicated by the low error rate in Figure 3a. Unsurprisingly, we can approximate INTENSITY policies from a very small database D built from other INTENSITY policies.

LOCATION **Policy.** Figure 3b plots the visual fidelity error when simulating a LOCATION policy from the database of INTENSITY policy trajectories. When D is small, the error is very high. MFMC is unable to reduce this error as D grows, because its distance metric does not find matching fire conditions for similar landscapes. In contrast, because the MFMCi methods are matching on the smaller Markov state variables, they are able to find good matching trajectories. The debiased version of MFMCi outperforms the biased version for the smaller database sizes. In the biased version the matching operation repeatedly stitches over long distances to find a database trajectory with a matching action. Debiased MFMCi avoids this mistake. This explains why debiased MFMCi rapidly decreases the error while biased MFMCi takes a bit longer but then catches up at roughly $|D| =$40,000.

FUEL **Policy.** The FUEL policy shows a best case scenario for the biased database. Within 7 time steps, fuel accumulation causes the policy action to switch from let-burn-all to suppress-all. Since all of the trajectories in the database have a consistent probability of suppressing fires throughout all 100 years, the ideal algorithm will select a trajectory that allows all wildfires to burn for 7 years (to reduce fuel accumulation), then stitch to the most similar trajectory in year 8 that will suppress all fires. The biased database will perform this “policy switching” by jumping between trajectories until the current trajectory always suppresses the fire.

In summary, our experiments show that MFMCi is able to generalize across policy classes and that it requires only a small number of database trajectories to accurately reproduce the median of each state variable at each future time step. In general, it appears to be better to create a debiased database than a biased database having the same number of tuples.

Our results allow our forestry stakeholders to interactively explore a range of policies within a visual analytic system. We further show how these results support interactive policy optimization for user-specified policy functions in McGregor et al. (2017b).

References

[Bellman1957] Bellman, R. 1957. *Dynamic Programming*. New Jersey: Princeton University Press.

[Dixon2002] Dixon, G. 2002. *Essential FVS : A User's Guide to the Forest Vegetation Simulator*. Number November 2015. Fort Collins, CO: USDA Forest Service.

[Finney1998] Finney, M. A. 1998. *FARSITE: fire area simulator model development and evaluation*. Missoula, MT: USDA Forest Service, Rocky Mountain Research Station.

[Fonteneau et al.2010] Fonteneau, R.; Murphy, S. A.; Wehenkel, L.; and Ernst, D. 2010. Model-Free Monte Carlo-like Policy Evaluation. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)* 217–224.

[Fonteneau et al.2013] Fonteneau, R.; Murphy, S. a.; Wehenkel, L.; and Ernst, D. 2013. Batch Mode Reinforcement Learning based on the Synthesis of Artificial Trajectories. *Annals of Operations Research* 208(1):383–416.

[Houtman et al.2013] Houtman, R. M.; Montgomery, C. A.; Gagnon, A. R.; Calkin, D. E.; Dietterich, T. G.; McGregor, S.; and Crowley, M. 2013. Allowing a Wildfire to Burn: Estimating the Effect on Future Fire Suppression Costs. *International Journal of Wildland Fire* 22(7):871–882.

[McGregor et al.2017a] McGregor, S.; Houtman, R.; Montgomery, C.; Metoyer, R.; and Dietterich, T. G. 2017. Factoring Exogenous State for Model-Free Monte Carlo. *arXiv preprint arXiv:1703.09390*.

[McGregor et al.2017b] McGregor, S.; Houtman, R.; Montgomery, C.; Metoyer, R.; and Dietterich, T. G. 2017. Fast Optimization of Wildfire Suppression Policies with SMAC. *arXiv preprint arXiv:1703.09391*.

[McGregor et al.2015] McGregor, S.; Buckingham, H.; Dietterich, T. G.; Houtman, R.; Montgomery, C.; and Metoyer, R. 2015. Facilitating Testing and Debugging of Markov Decision Processes with Interactive Visualization. In *IEEE Symposium on Visual Languages and Human-Centric Computing*.

[Puterman1994] Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1st edition.

Mellowmax: An Alternative Softmax Operator for Reinforcement Learning

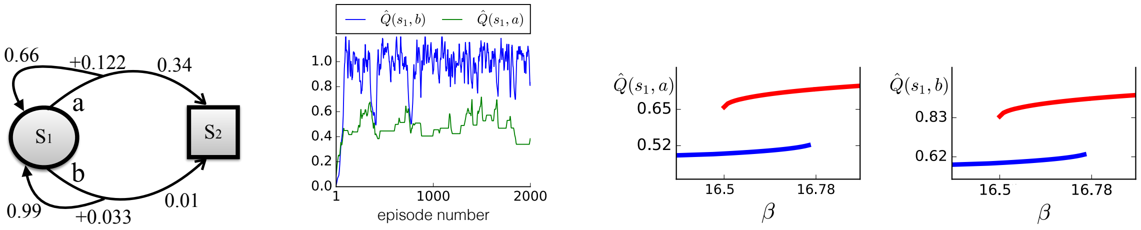
Kavosh Asadi
Brown University
kavosh@brown.edu

Michael L. Littman
Brown University
mlittman@brown.cs.edu

Abstract

A softmax operator applied to a set of values acts somewhat like the maximization function and somewhat like an average. In sequential decision making, softmax is often used in settings where it is necessary to maximize utility but also to hedge against problems that arise from putting all of one's weight behind a single maximum utility decision. The Boltzmann softmax operator is the most commonly used softmax operator in this setting, but we show that this operator is prone to misbehavior. In this work, we study and evaluate an alternative softmax operator that, among other properties, is both a non-expansion (ensuring convergent behavior in learning and planning) and differentiable (making it possible to improve decisions via gradient descent methods).

Keywords: Exploration-Exploitation, Convergence, Softmax, Convexity



(a) (b) (c)

Figure 1: A simple MDP with two states, two actions, and $\gamma = 0.98$. The use of SARSA with Boltzmann softmax policy is not sound in this simple domain.

1 Introduction

There is a fundamental tension in decision making between choosing the action that has highest expected utility and avoiding "starving" the other actions. The issue arises in the context of the exploration–exploitation dilemma (Thrun, 1992), non-stationary decision problems (Sutton, 1990), and when interpreting observed decisions (Baker et al., 2007).

In on-policy reinforcement learning, a typical approach to addressing this tension is the use of *softmax* operators for value-function optimization, and softmax policies for action selection. Examples include on-policy value-function-based methods such as SARSA (Rummery & Niranjan, 1994) or expected SARSA (Sutton & Barto, 1998; Van Seijen et al., 2009), and policy-search methods such as REINFORCE (Williams, 1992).

An ideal softmax operator is a parameterized set of operators that:

1. has parameter settings that allow it to approximate maximization arbitrarily accurately to perform reward-seeking behavior
2. is a non-expansion for all parameter settings ensuring convergence to a unique fixed point;
3. is differentiable to make it possible to improve via gradient descent; and
4. puts non-zero weights on non-maximizing actions to avoid the starvation of these actions.

Specifically, the Boltzmann softmax operator $\text{boltz}_\beta(\mathbf{X}) = \frac{\sum_{i=1}^{n} x_i \, e^{\beta x_i}}{\sum_{i=1}^{n} e^{\beta x_i}}$, is not a non-expansion. It is, therefore, prone to misbehavior as will be shown in the next section.

In the following section, we provide a simple example illustrating why the non-expansion property is important, especially in the context of planning and on-policy learning. We then present a new softmax operator that is similar to the Boltzmann operator yet is a non-expansion. We prove several critical properties of this new operator and then present empirical results.

2 Boltzmann Misbehaves

We first show that boltz_β can lead to problematic behavior. To this end, we ran SARSA with Boltzmann softmax policy on the MDP shown on Figure 1(a). The use of Boltzmann softmax policy in SARSA results in an unstable action-value estimation, as shown in Figure 1(b). As shown in Figure 1(c), there are two distinct fixed points for $\beta \in [16.55, 16.78]$. The behavior of SARSA with Boltzmann softmax policy, therefore, results from the algorithm stochastically bouncing back and forth between the two fixed points.

SARSA is known to converge in the tabular setting using ϵ-greedy exploration (Littman & Szepesvári, 1996), under decreasing exploration (Singh et al., 2000), and to a region in the function-approximation setting (Gordon, 2001). There are also variants of the SARSA update rule that converge more generally (Perkins & Precup, 2002; Baird & Moore, 1999; Van Seijen et al., 2009). However, this example is the first, to our knowledge, to show that SARSA fails to converge in the tabular setting with Boltzmann softmax.

3 Mellowmax Operator

We advocate for an alternative softmax operator defined as follows:

$$\text{mm}_\omega(\mathbf{X}) = \frac{\log(\frac{1}{n}\sum_{i=1}^{n} e^{\omega x_i})}{\omega} \,,$$

which can be viewed as a particular instantiation of the quasi-arithmetic mean (Beliakov et al., 2016). It can also be derived from information theoretical principles as a way of regularizing policies toward the uniform distribution with a cost function defined by KL divergence (Todorov, 2006; Fox et al., 2015).

We can show that $\mathrm{mm}_\omega(\mathbf{X})$, which we refer to as *mellowmax*, has all four of the desired properties. We prove that mm_ω is a non-expansion (Property 2), and therefore, SARSA under mm_ω is guaranteed to converge to a unique fixed point. The proofs for other properties are left to the reader.

Let $\mathbf{X} = x_1, \dots, x_n$ and $\mathbf{Y} = y_1, \dots, y_n$ be two vectors of values. Let $\Delta_i = x_i - y_i$ for $i \in \{1, \dots, n\}$ be the difference of the ith components of the two vectors. Also, let i^* be the index with the maximum component-wise difference, $i^* = \mathrm{argmax}_i \, \Delta_i$. For simplicity, we assume that i^* is unique. Also, without loss of generality, we assume that $x_{i^*} - y_{i^*} \geq 0$. It follows that:

$$
\begin{aligned}
&\big|\mathrm{mm}_\omega(\mathbf{X}) - \mathrm{mm}_\omega(\mathbf{Y})\big| \\
=\; & \Big|\log\big(\frac{1}{n}\sum_{i=1}^{n} e^{\omega x_i}\big)/\omega \; - \log\big(\frac{1}{n}\sum_{i=1}^{n} e^{\omega y_i}\big)/\omega \Big| \\
=\; & \Big|\log \frac{\sum_{i=1}^{n} e^{\omega\left(y_i + \Delta_i\right)}}{\sum_{i=1}^{n} e^{\omega y_i}}/\omega \Big| \\
\leq\; & \Big|\log \frac{\sum_{i=1}^{n} e^{\omega\left(y_i + \Delta_{i^*}\right)}}{\sum_{i=1}^{n} e^{\omega y_i}}/\omega \Big| \\
=\; & \Big|\log \frac{e^{\omega\Delta_{i^*}}\sum_{i=1}^{n} e^{\omega y_i}}{\sum_{i=1}^{n} e^{\omega y_i}}/\omega \Big| \\
=\; & \big|\log(e^{\omega\Delta_{i^*}})/\omega\big| = \big|\Delta_{i^*}\big| = \max_i \big|x_i - y_i\big| \,,
\end{aligned}
$$

allowing us to conclude that mellowmax is a non-expansion.

4 Maximum Entropy Mellowmax Policy

As described, mm_ω computes a value for a list of numbers. However, it is often useful to actually provide a probability distribution over the actions such that (1) a non-zero probability mass is assigned to each action, and (2) the resulting expected value equals the computed value. We address this as a maximum entropy problem (Cover & Thomas, 2006; Peters et al., 2010). We formally define the maximum entropy mellowmax policy of a state s as:

$$
\pi_{\mathrm{mm}}(s) = \underset{\pi}{\mathrm{argmin}} \sum_{a\in\mathcal{A}} \pi(a|s)\log\big(\pi(a|s)\big) \tag{1}
$$

$$
\text{subject to} \begin{cases} \sum_{a\in\mathcal{A}} \pi(a|s)\hat{Q}(s,a) = \mathrm{mm}_\omega(\hat{Q}(s,.)) \\ \pi(a|s) \geq 0 \\ \sum_{a\in\mathcal{A}} \pi(a|s) = 1 \,. \end{cases}
$$

Note that this optimization problem is convex and can be solved reliably using any numerical convex optimization library.

Another way of looking for the solution is to use the method of Lagrange multipliers. The solution in this case, which we refer to as maximum entropy mellowmax policy, has the form:

$$
\pi_{mm}(a|s) = \frac{e^{\beta\hat{Q}(s,a)}}{\sum_{a\in\mathcal{A}} e^{\beta\hat{Q}(s,a)}} \quad \forall a \in \mathcal{A} \,,
$$

where β is the solution to the following:

$$
\sum_{a\in\mathcal{A}} e^{\beta\Big(\big(\hat{Q}(s,a) - \mathrm{mm}_\omega\big(\hat{Q}(s,.)\big)\Big)}\Big(\hat{Q}(s,a) - \mathrm{mm}_\omega\big(\hat{Q}(s,.)\big)\Big) = 0
$$

The solution can be found easily using any root-finding algorithm. It is also simple to show that a unique root always exists.

This policy has the same form as Boltzmann softmax, but with a parameter β whose value depends indirectly on ω. This mathematical form arose not from the structure of mm_ω, but from maximizing the entropy. One way to view the use of the mellowmax operator, then, is as a form of Boltzmann softmax policy with a parameter chosen adaptively to ensure that the non-expansion property holds.

5 Experiments On MDPs

5.1 Random MDPs

The example in Figure 1 was created carefully by hand. It is interesting to know whether such examples are likely to be encountered naturally. To this end, we constructed 200 MDPs as follows: We sampled $|\mathcal{S}|$ from $\{2, 3, ..., 10\}$ and $|\mathcal{A}|$ from $\{2, 3, 4, 5\}$ uniformly at

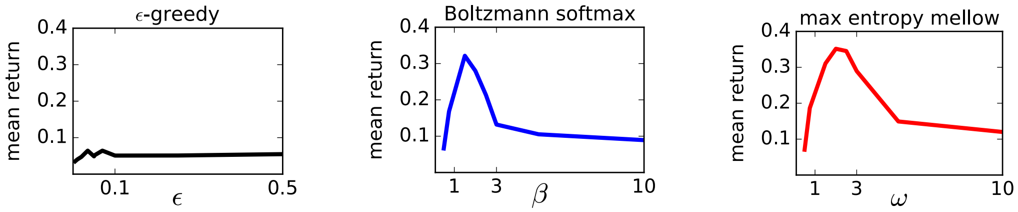


Figure 2: A simple MDP with two states, two actions, and $\gamma = 0.98$. The use of SARSA with Boltzmann softmax policy is not sound in this simple domain.

random. We initialized the transition probabilities by sampling uniformly from $[0, .01]$. We then added to each entry, with probability 0.5, Gaussian noise with mean 1 and variance 0.1. We next added, with probability 0.1, Gaussian noise with mean 100 and variance 1. Finally, we normalized the raw values to ensure that we get a transition matrix. We did a similar process for rewards, with the difference that we divided each entry by the maximum entry and multiplied by 0.5 to ensure that $R_{\max} = 0.5$.

We measured the failure rate of GVI under boltz$_\beta$ and mm$_\omega$ by stopping GVI when it did not terminate in 1000 iterations. We also computed the average number of iterations needed before termination. A summary of results is presented in the table below. Mellowmax outperforms Boltzmann based on the three measures provided in the table.

	MDPs, no terminate	MDPs, >1 fixed points	average iterations
boltz$_\beta$	8 of 200	3 of 200	231.65
mm$_\omega$	**0**	**0**	**201.32**

5.2 Multi-passenger Taxi Domain

We evaluated SARSA on the multi-passenger taxi domain introduced by Dearden et al. (1998). One challenging aspect of this domain is that it admits many locally optimal policies. Note also that Boltzmann softmax performs remarkably well on this domain, outperforming sophisticated Bayesian reinforcement-learning algorithms (Dearden et al., 1998). As shown in Figure 2, Maximum entropy mellowmax policy is no worse than Boltzmann softmax, here, suggesting that the greater stability does not come at the expense of less effective exploration.

6 Related Work

Softmax operators play an important role in sequential decision-making algorithms. In model-free reinforcement learning, they can help strike a balance between exploration (mean) and exploitation (max). Decision rules based on epsilon-greedy and Boltzmann softmax, while very simple, often perform surprisingly well in practice, even outperforming more advanced exploration techniques (Kuleshov & Precup, 2014) that require significant approximation for complex domains. When learning "on policy", exploration can (Rummery & Niranjan, 1994) and perhaps should (John, 1994) become part of the value-estimation process itself.

Analyses of the behavior of human subjects in choice experiments very frequently use softmax as it forms an important part of the most accurate predictor of human decisions in normal-form games (Wright & Leyton-Brown, 2010), quantal level-k reasoning (QLk). Softmax-based fixed points play a crucial role in this work. As such, mellowmax could potentially make a good replacement resulting in better behaved solutions.

7 Conclusion and Future Work

We proposed the mellowmax operator as an alternative to the Boltzmann softmax operator. We showed that mellowmax is a non-expansion, unlike Boltzmann, and that it works favorably in practice. Arguably, mellowmax could be used in place of Boltzmann throughout reinforcement-learning research.

A future direction is to analyze the fixed point of planning, reinforcement-learning, and game-playing algorithms when using Boltzmann and mellowmax operators. An interesting analysis could be one that bounds the suboptimality of fixed points found by value iteration.

8 Acknowledgments

The authors gratefully acknowledge the assistance of George Konidaris.

References

Baird, Leemon and Moore, Andrew W. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 968–974, 1999.

Baker, Chris L, Tenenbaum, Joshua B, and Saxe, Rebecca R. Goal inference as inverse planning. In *Proceedings of the 29th Annual Meeting of the Cognitive Science Society*, 2007.

Beliakov, Gleb, Sola, Humberto Bustince, and Sánchez, Tomasa Calvo. *A Practical Guide to Averaging Functions*. Springer, 2016.

Cover, T.M. and Thomas, J.A. *Elements of Information Theory*. John Wiley and Sons, 2006.

Dearden, Richard, Friedman, Nir, and Russell, Stuart. Bayesian Q-learning. In *Fifteenth National Conference on Artificial Intelligence (AAAI)*, pp. 761–768, 1998.

Fox, Roy, Pakman, Ari, and Tishby, Naftali. Taming the noise in reinforcement learning via soft updates. *arXiv preprint arXiv:1512.08562*, 2015.

Gordon, Geoffrey J. Reinforcement learning with function approximation converges to a region, 2001. Unpublished.

John, George H. When the best move isn't optimal: Q-learning with exploration. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 1464, Seattle, WA, 1994.

Kuleshov, Volodymyr and Precup, Doina. Algorithms for multi-armed bandit problems. arXiv preprint arXiv:1402.6028, 2014.

Littman, Michael L. and Szepesvári, Csaba. A generalized reinforcement-learning model: Convergence and applications. In Saitta, Lorenza (ed.), *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 310–318, 1996.

Perkins, Theodore J and Precup, Doina. A convergent form of approximate policy iteration. In *Advances in Neural Information Processing Systems*, pp. 1595–1602, 2002.

Peters, Jan, Mülling, Katharina, and Altun, Yasemin. Relative entropy policy search. In *AAAI*. Atlanta, 2010.

Rummery, G. A. and Niranjan, M. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.

Singh, Satinder, Jaakkola, Tommi, Littman, Michael L., and Szepesvári, Csaba. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 39:287–308, 2000.

Sutton, Richard S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pp. 216–224, Austin, TX, 1990. Morgan Kaufmann.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

Thrun, Sebastian B. The role of exploration in learning control. In White, David A. and Sofge, Donald A. (eds.), *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pp. 527–559. Van Nostrand Reinhold, New York, NY, 1992.

Todorov, Emanuel. Linearly-solvable markov decision problems. In *NIPS*, pp. 1369–1376, 2006.

Van Seijen, Harm, Van Hasselt, Hado, Whiteson, Shimon, and Wiering, Marco. A theoretical and empirical analysis of expected sarsa. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 177–184. IEEE, 2009.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8 (3):229–256, 1992.

Wright, James R. and Leyton-Brown, Kevin. Beyond equilibrium: Predicting human behavior in normal-form games. In *AAAI*, 2010.

Mechanisms of Overharvesting in Patch Foraging

Gary A. Kane
Department of Psychology
Princeton University
Princeton, NJ 08540
gkane@princeton.edu

Aaron M. Bornstein
Neuroscience Institute
Princeton University
aaronmb@princeton.edu

Amitai Shenhav
Department of Cognitive, Linguistic,
and Psychological Sciences
Brown University
amitai_shenhav@brown.edu

Robert C. Wilson
Department of Psychology
University of Arizona
bob@email.arizona.edu

Nathaniel D. Daw
Department of Psychology
Princeton University
ndaw@princeton.edu

Jonathan D. Cohen
Department of Psychology
Princeton University
jdc@princeton.edu

Abstract

Serial stay-or-search decisions are ubiquitous across many domains, including decisions regarding employment, relationships, and foraging for resources or information. Studies of animal foraging, in which animals decide to harvest depleting rewards contained within a patch or to leave the patch in search of a new, full one, have revealed a consistent bias towards overharvesting, or staying in patches longer than is predicted by optimal foraging theory (the Marginal Value Theorem; MVT) [1]. Yet, the cognitive biases that lead to overharvesting are poorly understood. We attempt to determine the cognitive biases that underlie overharvesting in rats. We characterized rat foraging behavior in response to two basic manipulations in patch foraging tasks: travel time between reward sources and depletion rate of the source; and to two novel manipulations to the foraging environment: proportional changes to the size of rewards and length of delays, and placement of delays (pre- vs. post-reward). In response to the basic manipulations, rats qualitatively followed predictions of MVT, but stayed in patches longer than is predicted. In the latter two manipulations, rats deviated from predictions of MVT, exhibiting changes in behavior not predicted by MVT. We formally tested whether four separate cognitive biases – subjective costs, decreasing marginal utility for reward, discounting of future reward, and ignoring post-reward delays – could explain overharvesting in the former two manipulations and deviations from MVT in the latter two. All the biases tested explained overharvesting behavior in the former contexts, but only one bias – in which rats ignore post-reward delays – also explained deviations from MVT in the latter contexts. Our results reveal that multiple cognitive biases may contribute to overharvesting, but inaccurate estimation of post-reward delays provided the best explanation across all contexts.

Keywords: foraging; subjective utility; delay discounting

Acknowledgements

This research was supported by NIMH grant F31MH109286 (GAK).

1 Introduction

Serial stay-or-search decisions, ones in which an agent must decide to stick with a current resource or to leave it behind to search for something new, are encountered in many domains, including decisions regarding employment, relationships, and animal foraging. In animal foraging, specifically patch foraging, animals repeatedly decide to harvest a depleting reward within a patch, or to search for a new, likely richer one, which comes at the cost of time and/or effort spent travelling to the new patch. The optimal solution in patch foraging problems is given by the Marginal Value Theorem (MVT) [1]: leave when the local reward rate within a patch depletes below the global average reward rate for the environment. MVT makes two main predictions: i) in patches that contain more reward than average, stay longer to exploit such reward, and ii) when the cost of searching for a new patch is greater (e.g. the time or effort required to travel to a new patch is greater), stay longer in all patches. Many animals qualitatively follow predictions of MVT [2]. However, in most tests, animals tend to overharvest, or stay in patches longer than is predicted by MVT [2–5].

Hypotheses to explain overharvesting include common biases in intertemporal choice, such as i) subjective costs, like an aversion to leaving the patch [6, 7]; ii) decreasing marginal utility, in which large rewards available in a new patch are not viewed as proportionally larger than the smaller, depleted rewards available in the current patch [3]; iii) discounting future rewards, in which the value of large rewards available in a new patch are discounted by virtue of being available later, above and beyond the time it takes to travel to the new patch [3, 6, 8]; and iv) ignoring post-reward delays, which causes overestimation of reward rate within the patch due to inaccurate estimation of the time taken to obtain reward [6, 8–11]. Although overharvesting is widely observed, there have been few direct investigations into the underlying mechanisms. We characterize rat foraging behavior in response to multiple manipulations in an operant chamber based patch foraging task, and formally test these hypotheses to rat foraging behavior.

2 Foraging Task

8 adult Long-Evans rats were used. On a series of trials performed in a standard operant chamber, rats decided to harvest sugar-water reward from a depleting patch by pressing a lever at the front of the chamber, or to travel to a new full, patch by nosepoking at the back of the chamber. To cue the beginning of a trial, lights above the lever and within the nosepoke illuminated, indicating that the rat could decide to harvest reward from the current patch (lever press) or to travel to a new patch (nosepoke). The time from start of the trial to the decision was recorded as decision time (DT). If the rat pressed the lever to harvest from the activated patch, a cue light turned on in the reward magazine next to the lever following a short handling time delay (HT, 0.5 s), and liquid sucrose was delivered when the rats head entered the magazine. An intertrial interval (ITI) began as soon as the rat entered the reward magazine. To control the reward rate within the patch, the length of the ITI was adjusted based on the DT of the current trial, such that the length of all lever press trials was equivalent. If the rat nosepoked to leave the patch, the lever retracted for a delay period, simulating the travel time, after which, the lever on the opposite side of the front of the chamber extended, representing a new patch from which the rat could harvest.

We performed four manipulations to the foraging environment in four separate experiments. In each experiment, rats were tested on two conditions in a block design, counterbalanced. In the first experiment, within each condition, rats encountered three different patch types, which started with varying amount of reward (60, 90, or 120 microliters (μL)), each depleting at the same rate (8 μL per trial). Between conditions, rats were tested on either a 10 s or 30 s travel time delay. In the next three experiments, rats encountered a single patch type within each condition. Second, rats were tested on two different depletion rates (8 or 16 μL per harvest). Next, rats were tested on two environments with equivalent reward rates, but one environment provided larger rewards with longer delays (starting volume = 90/180 μL, depletion rate = 8/16 μL, time of trial = 10/20 s; referred to as the scale experiment). Last, rats were tested in two environments in which the time for each trial and reward rates were equivalent across conditions, but in one condition rats experienced a longer handling time delay but shorter ITI (HT = 0.5/3.5s with ITI = 10/7 s; referred to as the delay experiment). T-test and ANOVA were used to compare rat behavior between conditions.

Table 1: Description of timing variables

Abbrev.	Variable	Description
DT	Decision time	Latency from start of trial to lever press or nosepoke
HT	Handling time	Latency from lever press to reward
ITI	Intertrial interval	Delay from reward to start of next trial
τ	Travel time	Time from nosepoke to start of next trial, in new patch

3 Models of Foraging Behavior

All models were constructed as Markov Decision Processes. States were represented as trials within patches. A decision to stay in the patch (i.e. harvest from the patch) provided reward for staying in state s, $r_{stay,s}$, and caused transition to state $s+1$. A decision to leave the patch resulted in travel time delay, τ, followed by the first reward in the next patch, r_{leave}, and associated ITI following the reward, ITI_{leave}. Three of the models were based on MVT: one incorporating a constant subjective cost (subjective cost), one that accounted for diminishing marginal returns for larger rewards (subjective utility), and one ignoring post-reward delays. A fourth model — a delay discounting model — was not based on MVT.

For each of the MVT models, the value of staying in state s, $V_{stay,s}$, was the local reward rate within the patch,

$$T_{stay,s} = DT_{stay,s} + HT + ITI_{stay,s}$$

$$V_{stay,s} = \frac{r_{stay,s}}{T_{stay,s}}$$

and the value of leaving the patch in state s, $V_{leave,s}$, was the global reward rate across patches,

$$T_{leave} = DT_{leave} + \tau + ITI_{leave}$$

$$R_s = p_{stay,s} \cdot r_{stay,s} + p_{leave,s} \cdot r_{leave}$$

$$T_s = p_{stay,s} \cdot T_{stay,s} + p_{leave,s} \cdot T_{leave}$$

$$V_{leave} = \frac{\sum_s p_s \cdot R_s}{\sum_s p_s \cdot T_s}$$

where R_s and T_s were the average reward and average time for state s, $p_{stay,s}$ was the probability of choosing to stay in state s, and p_s was the probability of being in state s. Optimal behavior was to leave the patch when $V_{leave} >= V_{stay,s}$ (i.e. when the global average reward rate is greater than the local reward rate in the patch). To model rats behavior, patch leaving distributions were assumed to be normally distributed with respect to $V_{leave} - V_{stay,s}$, with mean $= 0$ (i.e. $V_{leave} = V_{stay,s}$) and variance σ, a free parameter.

To account for subjective costs, a constant, c, representing an aversion to leaving the patch, was added to the model, such that the patch leaving distribution was normally distributed with respect to $V_{leave} - V_{stay,s} - c$. This subjective cost reduced the reward rate threshold to leave patches.

For the subjective utility model, the utility for taking action a in state s increased monotonically but sublinear to the size of the reward according to a power utility function, dependent on a free parameter, η: $u_{a,s} = r_{a,s}^{\eta}$. $V_{stay,s}$ and V_{leave} were calculated in terms of utility rather than reward. Under this hypothesis, larger rewards available in a new patch are not valued as proportionally larger to the smaller rewards available in the current patch, resulting in a reduced estimate of the global average reward rate, reducing the reward rate threshold to leave patches.

For the ignoring post-reward delays model, delays that occur after receiving reward, but before a decision was made on the next trial (e.g. ITI after reward and DT prior to making next decision), T_{post}, were treated differently than time delays that occurred between the decision and receiving a reward (e.g. HT between lever press and reward, or travel time delay between nosepoke and first reward in the next patch). We tested multiple functions for how post-reward delays might have been treated, all in which the perceived time increased monotonically but sublinear to actual time, including a linear function with slope < 1, a power function, and an exponential function. The exponential function provided the best fit across all experiments, and was used for further testing: $T_{post} = \dfrac{1 - e^{-\alpha(DT_{a,s} + ITI_{a,s})}}{\alpha}$. $V_{stay,s}$ and V_{leave} were calculated using T_{post} in place of DT and ITI. By ignoring, or underestimating, the time between receiving reward and making the next decision to stay vs. leave, the local reward rate is overestimated. Consequently, it will take longer for the reward rate in the patch to deplete to global average reward rate.

The delay discounting model was simlar to MVT, in that it optimizes future rewards, but with less weight to rewards tahat occur further into the future:

$$d(t, k) = 1/(1 + kt)$$

$$V_{stay,s} = d(HT, k) \left[r_{stay,s} + d(ITI_{stay,s}, k) \sum_{s'} p(s'|stay, s) \sum_{s'} p_{a,s'} V_{a,s'} \right]$$

$$V_{leave} = d(\tau, k) \left[r_{leave} + d(ITI_{leave}, k) \sum_{s'} p(s'|leave) \sum_{s'} p_{a,s'} V_{a,s'} \right]$$

where $d(t, k)$ was a hyperbolic discount function of time t, with a free parameter, k. $p(s|a, s)$ was the conditional probability of being in future state s given action a was taken in state s, $p_{a,s}$ was the probability of taking action a in future state s, and $V_{a,s}$ was the value of taking action a in future state s. As the discount parameter, k, approached zero (no discounting of future reward), this model converged to MVT; that is, it sought to maximize all future reward. As k increased, future rewards were discounted, such that i) the value of large rewards in a new patch are discounted above and beyond the travel time between patches, and ii) the model sought to maximize reward into the future, but over shorter periods of time. Since rewards obtained immediately are valued greater than those that come following longer delays (e.g. the travel time), the value for staying in the patch will be greater relative to the value of leaving the patch.

For all models, one set of parameters was fit to each animal per experiment, to maximize the likelihood of the data from both conditions in that experiment.

4 Behavioral Results

In the travel time experiment, longer travel time between patches reduced the global average reward rate, thus MVT predicted that rats should stay longer in all patches when travel time is longer. Consistent with this prediction, rats stayed for more trials in all patches when travel time was longer ($F(1,7) = 71.4, p < .001$; Figure 1A). Also as predicted by MVT, rats stayed longer in patches that started with larger reward volume ($F(2,14) = 3145, p < .001$). However, rats overharvested, staying in all patches longer than is predicted by MVT.

With quicker reward depletion, the local reward rate depleted to the global average reward rate quicker, so MVT predicted earlier patch leaving. Rats left patches earlier when they depleted more quickly ($t(7) = 15.835, p < .001$; Figure 1B), but again, rats overharvested, staying in patches longer than is predicted by MVT.

In the scale experiment (larger rewards and longer delays), the trial-by-trial local reward rate and the global average reward rate were equivalent across conditions, and MVT predicted no change in behavior. However, rats stayed in patches longer in the environment that offered larger rewards with longer delays ($t(7) = 10.039, p < .001$; Figure 1C).

Lastly, in the delay experiment, the total time for each trial was equivalent across conditions, so MVT did not predict any change in behavior. Yet, rats left patches earlier in the environment with a longer pre-reward delay and shorter post-reward delay ($t(7) = 7.453, p < .001$; Figure 1D).

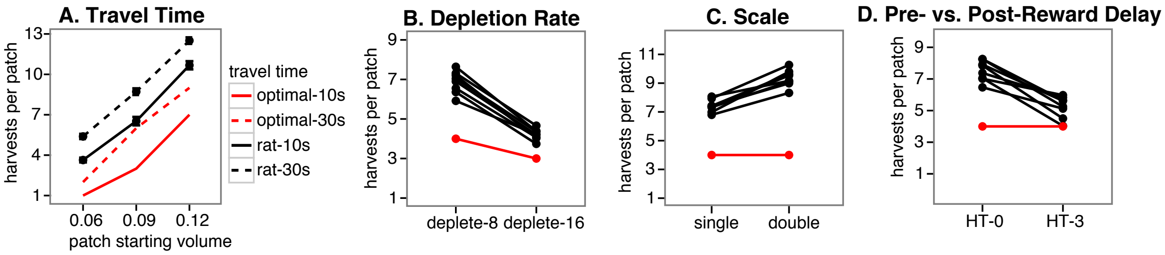


Figure 1: Behavior in the foraging task. Rat behavior is indicated in black points and lines, MVT optimal behavior in red. A) Points and error bars are mean ± standard error. B-D) Each point and line represents behavior from an individual rat.

5 Model Results

Predictions of the best fitting models for each experiment are presented in Figure 2. The subjective cost model produced a predicted number of harvests per patch similar to that exhibited by rats in the travel time and depletion rate experiments. As this model only allows for a constant change in the reward rate threshold to leave patches, it is unlikely to account for behavior in which rats select a different threshold between conditions, as was the case in the scale and delay experiments. In both of these experiments, the subjective cost model failed to explain the change in behavior across conditions.

Similarly, the subjective utility model explained overharvesting in the travel time and depletion rate experiments. For the scale experiment, nonlinear perception of reward provided the possibility that this model could predict greater overharvesting in environments with larger rewards, despite equivalent reward rates. However, this model failed to explain later patch leaving in the scale experiment. As this model is indifferent to the placement of delays, it also failed to explain earlier patch leaving due to the introduction of a pre-reward delay.

The delay discounting model also accurately predicted overharvesting in the travel time and depletion rate experiments. In the scale experiment, longer delays resulted in greater discounting of the larger rewards, and this model predicted later patch leaving when given larger rewards with longer delays. In the delay experiment, qualitatively, the value of reward within the patch could be reduced by the introduction of a pre-reward delay, thus predicting earlier patch leaving in this condition. However, quantitatively, this model did not predict earlier patch leaving due to a short pre-reward delay.

Lastly, as did the other three models, the ignore post-reward delays model accurately predicted overharvesting in the travel time and depletion rate experiments. Furthermore, in the scale experiment, longer post-reward delays resulted in greater overestimation of reward rate in the condition which provided larger rewards, and the model accurately predicted rats woudl spend more time in patches in this condition. In the delay experiment, the condition with a longer pre-reward delay also had a shorter post-reward delay. The shorter post-reward delay resulted in less overestimation of reward rate, and the model predicted earlier patch leaving. This model explained rat foraging behavior in all four experiments.

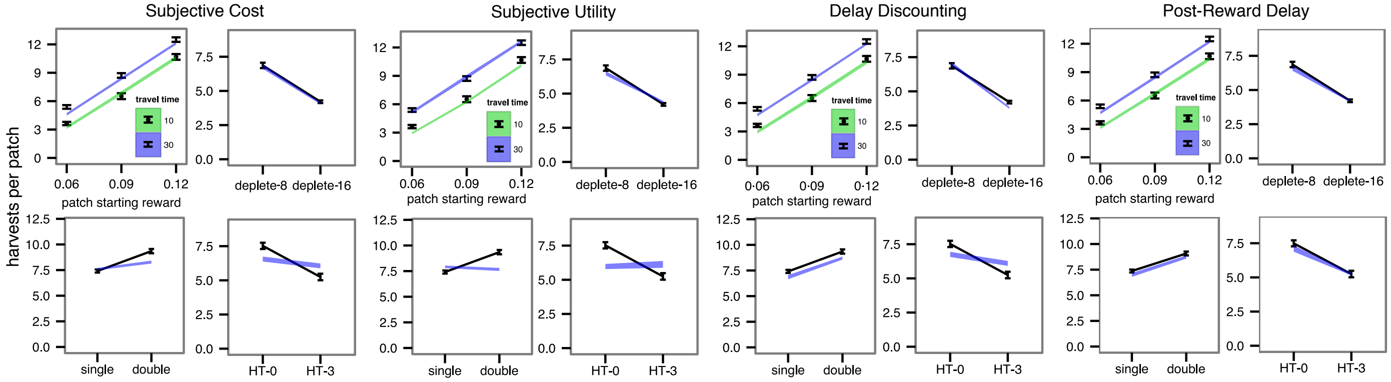


Figure 2: Model predictions for each of the four models for each experiment. For each model, data from the travel time experiment is in top left panel, depletion rate in top right, scale of reward and delay in bottom left, and pre- vs. post-reward delay in bottom right. Black points and errorbars represent mean ± standard error. Colored lines indicate model predictions; the thickness of the line indicates standard error of model predictions.

6 Conclusions

Rats, like other animals, qualitatively followed basic predictions of MVT: stay longer in patches with longer travel time, and leave patches earlier when they deplete quicker. However, rats deviated from MVT in specific cases: rats were sensitive to the scale of reward and delay, and they were sensitive to the placement of delays. We found that all four cognitive biases tested – subjective costs, diminishing marginal utility, discounting of future rewards, and ignoring post-reward delays – explained overharvesting in general, but only one bias, ignoring post-reward delays, could explain both overharvesting as well as deviations from MVT in all contexts tested. These results suggest that foraging behavior may be the result of multiple cognitive biases, but that inaccurate estimation of post-reward delays provided the best explanation across all contexts tested.

References

[1] Charnov, E. L. (1976) Optimal foraging, the marginal value theorem. *Theoretical Population Biology* **9**, 129–136.

[2] Stephens, D. W & Krebs, J. R. (1986) *Foraging Theory*. Vol. 121, p. 247.

[3] Constantino, S. M & Daw, N. D. (2015) Learning the opportunity cost of time in a patch-foraging task. *Cognitive, Affective, & Behavioral Neuroscience*.

[4] Hayden, B. Y, Pearson, J. M, & Platt, M. L. (2011) Neuronal basis of sequential foraging decisions in a patchy environment. *Nature Neuroscience* **14**, 933–939.

[5] Nonacs, P. (1991) State dependent behavior and the Marginal Value Theorem. *Behavioral Ecology* **12**, 71–83.

[6] Carter, E. C & Redish, A. D. (2016) Rats value time differently on equivalent foraging and delay-discounting tasks. *Journal of Experimental Psychology: General* **145**, 1093–1101.

[7] Wikenheiser, A. M, Stephens, D. W, & Redish, a. D. (2013) Subjective costs drive overly patient foraging strategies in rats on an intertemporal foraging task. *Proceedings of the National Academy of Sciences* **110**, 8308–13.

[8] Blanchard, T. C, Pearson, J. M, & Hayden, B. Y. (2013) Postreward delays and systematic biases in measures of animal temporal discounting. *Proceedings of the National Academy of Sciences* **110**, 15491–6.

[9] Bateson, M & Kacelnik, a. (1996) Rate currencies and the foraging starling: The fallacy of the averages revisited. *Behavioral Ecology* **7**, 341–352.

[10] Gallistel, C. R & Gibbon, J. (2000) Time, rate, and conditioning. *Psychological Review* **107**, 289–344.

[11] Kacelnik, A. (1997) in *Characterizing Human Psychological Adaptations*. Vol. 208, pp. 51–66.

Stochastic Primal-Dual Methods and Sample Complexity of Markov Decision Processes

Yichen Chen
Department of Computer Science
Princeton University
Princeton, NJ 08544
`yichenc@cs.princeton.edu`

Mengdi Wang
Department of Operation Research and Financial Engineering
Princeton University
Princeton, NJ 08544
`mengdiw@princeton.edu`

Abstract

We study the online estimation of the optimal policy of a Markov decision process (MDP). We propose a class of Stochastic Primal-Dual (SPD) methods which exploit the inherent minimax duality of Bellman equations. The SPD methods update a few coordinates of the value and policy estimates as a new state transition is observed. These methods use $\mathcal{O}|\mathcal{S}||\mathcal{A}|$ space and has low computational complexity per iteration. We first consider a basic version of SPD that uses Euclidean projection for both the primal and dual updates. We show that it find an ϵ-optimal policy regardless of the initial state, with high probability, using $\mathcal{O}\left(\frac{|\mathcal{S}|^4|\mathcal{A}|^2\sigma^2}{(1-\gamma)^6\epsilon^2}\right)$ iterations/samples for the infinite-horizon discounted-reward MDP and $\mathcal{O}\left(\frac{|\mathcal{S}|^4|\mathcal{A}|^2H^6\sigma^2}{\epsilon^2}\right)$ iterations/samples for the finite-horizon MDP. We then propose an accelerated version of SDP that uses relative entropy projection in the dual udate. We show that the improved SPD method achieves the sample/running-time complexity $\mathcal{O}\left(\frac{|\mathcal{S}|^3|\mathcal{A}|\log(|\mathcal{S}||\mathcal{A}|)\sigma^2}{(1-\gamma)^4\epsilon^2}\right)$ for the general discounted-reward MDPs. For MDPs that are "sufficiently" ergodic, the improved SPD has sample/running-time complexity $\mathcal{O}\left(\frac{|\mathcal{S}||\mathcal{A}|\log(|\mathcal{S}||\mathcal{A}|)\sigma^2}{(1-\gamma)^2\epsilon^2}\right)$.

Keywords: Markov decision process, stochastic algorithm, duality, sample complexity

1 Introduction

Markov decision process (MDP) is one of the most basic model of dynamic programming, stochastic control and Markov decision processes (MDPs); see the textbook references (Bertsekas, 2007; Bertsekas and Tsitsiklis, 1995; Sutton and Barto, 1998; Puterman, 2014). Given a controllable Markov chain and the distribution of state-to-state transition reward, the aim is to find the optimal action to perform at each state in order to maximize the expected overall reward. MDP and its numerous variants are widely applied in engineering systems, artificial intelligence, e-commerce and finance.

We study both the infinite-horizon discounted-reward MDP and the finite-horizon MDP. In both cases, we focus on MDP problems with a finite state space $\mathcal{S}$ and a finite action space $\mathcal{A}$. Suppose that we are given a simulation oracle that generates random state-to-state transitions and transitional rewards when an initial state and actions are specified. The algorithmic goal is to estimate the optimal policy of the unknown MDP based on empirical state transitions, without any prior knowledge or restrictive assumption about the underlying process.

We focus on the model-free learning setting where both transition probabilities and transitional rewards are unknown. By "model-free" we mean the desired algorithm does not explicitly estimate the transition probabilities of the MDP. We are interested in *model-free learning algorithm whose space complexity is sublinear to the size of the MDP model.* This is a critical difference between our proposed method with the upper-confidence reinforcement learning methods, which keep an estimate for each possible transition.

In the literature of approximate dynamic programming and reinforcement learning, many methods have been developed; recent examples including (Ortner and Auer, 2007; Dann and Brunskill, 2015; Lattimore et al., 2013; Lattimore and Hutter, 2012; Wang and Chen, 2016). Although researchers have made significant progress in the past decade, it remains unclear whether there is an approach that achieves both theoretical optimality as well as practical scalability (in space and running time). This is an active area of research.

In this abstract, we present a novel approach which is motivated by the linear programming formulation of the nonlinear Bellman equation, which is based on two recent manuscripts (Chen and Wang, 2016; Wang, 2017). We formulate the Bellman equation into a stochastic saddle point problem, where the optimal primal and dual solutions correspond to the optimal value and policy functions, respectively. We propose a class of Stochastic Primal-Dual algorithms (SPD) for the discounted MDP and the finite-horizon MDP. Each iteration of the algorithms updates the primal and dual solutions simultaneously using noisy partial derivatives of the Lagrangian function. In contrast to existing literatures where the learned policy is often inferred from the estimated action-value functions (Azar et al., 2011, 2012), the SPD methods *estimate the optimal policy directly*. The SPD methods are stochastic analogs of the primal-dual iteration for linear programming. They are crafted to take advantages of structures of the Bellman equation, and they also involve alternative projections onto specially constructed sets. The SPD methods are very easy to implement and exhibit favorable space complexity. To analyze its sample complexity, we adopt the notion of "Probably Approximately Correct" (PAC), which means to achieve an ϵ-optimal policy with high probability using sample size polynomial with respect to the problem parameters.

2 Stochastic Primal-Dual Methods

In this section, we present the class of stochastic primal-dual reinforcement learning methods. We will focus on the case of discounted Markov decision processes. The case of finite-horizon MDPs bear the same spirit and is omitted for brevity.

2.1 Preliminaries on MDP

We consider a discounted MDP described by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$ is a finite state space, $\mathcal{A}$ is a finite action space, $\gamma \in (0, 1)$ is a discount factor. If action a is selected while the system is in state i, the system transitions to state j with probability $P_a(i, j)$ and incurs a random reward $\hat{r}_{ija} \in [0, 1]$ with expectation r_{ija}.

Let $\pi : \mathcal{S} \mapsto \mathcal{A}$ be a *policy* that maps a state $i \in \mathcal{S}$ to an action $\pi(i) \in \mathcal{A}$. Consider the Markov chain under policy π. We denote its transition probability matrix by P_π and denote its transitional reward vector by r_π, i.e.,

$$P_\pi(i, j) = P_{\pi(i)}(i, j), \qquad r_\pi(i) = \sum_{j \in \mathcal{S}} P_{\pi(i)}(i, j) r_{ij\pi(i)}, \qquad i, j \in \mathcal{S}.$$

The objective is to find an optimal policy $\pi^* : \mathcal{S} \mapsto \mathcal{A}$ such that the infinite-horizon discounted reward is maximized, regardless of the initial state:

$$\max_{\pi : \mathcal{S} \mapsto \mathcal{A}} \mathbf{E}^\pi \left[\sum_{k=0}^{\infty} \gamma^k \hat{r}_{i_k i_{k+1} \pi(i_k)} \right],$$

where $\gamma \in (0,1)$ is a discount factor, $(i_0, i_1, \ldots)$ are state transitions generated by the Markov chain under policy π, and the expectation is taken over the entire process. We assume throughout that *there exists a unique optimal policy* π^* *to the MDP tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$. In other words, there exists one optimal action for each state.

We review the standard definitions of value functions.

Definition 1 *The value vector* $v^\pi \in \Re^{|\mathcal{S}|}$ *of a fixed policy* π *is defined as*

$$v^\pi(i) = \mathbf{E}^\pi \left[\sum_{k=0}^{\infty} \gamma^k \hat{r}_{i_k i_{k+1} \pi(i_k)} \,\middle|\, i_0 = i \right], \qquad i \in \mathcal{S}.$$

Definition 2 *The optimal value vector* $v^* \in \Re^{|\mathcal{S}|}$ *is defined as*

$$v^*(i) = \max_{\pi: \mathcal{S} \mapsto \mathcal{A}} \mathbf{E}^\pi \left[\sum_{k=0}^{\infty} \gamma^k \hat{r}_{i_k i_{k+1} \pi(i_k)} \,\middle|\, i_0 = i \right] = \mathbf{E}^{\pi^*} \left[\sum_{k=0}^{\infty} \gamma^k \hat{r}_{i_k i_{k+1} \pi^*(i_k)} \,\middle|\, i_0 = i \right], \qquad i \in \mathcal{S}.$$

For the sample complexity analysis of the proposed algorithm, we need a notion of sub-optimality of policies. We give its definition as below.

Definition 3 *We say that a policy* π *is* ϵ*-optimal if*

$$\max_{i \sim p_0} \mathbf{E} |v^\pi(i) - v^*(i)| \le \epsilon,$$

where p_0 is the initial distribution. We have also another notion of sub-optimality.

Definition 4 *We say that a policy* π *is absolute-*ϵ*-optimal if*

$$\max_{i} |v^\pi - v^*(i)| \le \epsilon,$$

If a policy is absolute-ϵ-optimal, it achieves ϵ-optimal reward regardless of the initial state distribution. We note that the absolute-ϵ-optimality is one of the strongest notions of sub-optimality for policies.

2.2 Linear Duality of The Bellman equation

According to the theory of dynamic programming (Bertsekas, 2007), a vector v^* is the optimal value function to the MDP if and only if it satisfies the following $|\mathcal{S}| \times |\mathcal{S}|$ system of equations, known as the *Bellman equation*, given by

$$v^*(i) = \max_{a \in \mathcal{A}} \left\{ \gamma \sum_{j \in \mathcal{S}} P_a(i,j) v^*(j) + \sum_{j \in \mathcal{S}} P_a(i,j) r_{ija} \right\}, \quad i \in \mathcal{S}, \tag{1}$$

When $\gamma \in (0,1)$, the Bellman equation has a unique fixed point solution v^*, and it equals to the optimal value function of the MDP. Moreover, a policy π^* is an optimal policy for the MDP if and only of it attains the minimization in the Bellman equation. Note that this is a *nonlinear* system of fixed point equations. Interestingly, the Bellman equation (1) is equivalent to the following $|\mathcal{S}| \times (|\mathcal{S}||\mathcal{A}|)$ linear programming (LP) problem (see Puterman (2014) Section 6.9. and the paper by de Farias and Van Roy (2003)):

$$\begin{aligned} &\text{minimize } \xi^T v \\ &\text{subject to } (\mathbf{I} - \gamma P_a) v - r_a \ge 0, \qquad a \in \mathcal{A}, \end{aligned} \tag{2}$$

where ξ is an arbitrary vector with positive entries, $P_a \in \Re^{|\mathcal{S}| \times |\mathcal{S}|}$ is matrix whose (i,j)-th entry equals to $P_a(i,j)$, $\mathbf{I}$ is the identity matrix with dimension $|\mathcal{S}| \times |\mathcal{S}|$ and $r_a \in \Re^{|\mathcal{S}|}$ is the expected state transition reward under action a given by $r_a(i) = \sum_{j \in \mathcal{S}} P_a(i,j) r_{ija}, i \in \mathcal{S}$.

We rewrite the LP program (2) into an equivalent minimax problem, given by

$$\min_{v \in \Re^{|\mathcal{S}|}} \max_{\lambda \ge 0} L(v, \lambda) = \xi^T v + \sum_{a \in \mathcal{A}} \lambda_a^T \left((\gamma P_a - \mathbf{I}) v + r_a \right). \tag{3}$$

The primal variable v is of dimension $|\mathcal{S}|$, and the dual variable

$$\lambda = (\lambda_a)_{a \in \mathcal{A}} = (\lambda_{a,i})_{a \in \mathcal{A}, i \in \mathcal{S}}$$

is of dimension $|\mathcal{S}| \cdot |\mathcal{A}|$. Each subvector $\lambda_a \in \Re^{|\mathcal{S}|}$ is the vector multiplier corresponding to constraint inequalities $(\mathbf{I} - \gamma P_a)\, v - r_a \geq 0$. Each entry $\lambda_{a,i} > 0$ is the scalar multiplier associated with the ith row of $(\mathbf{I} - \gamma P_a)\, v - r_a \geq 0$.

In order to develop an efficient algorithm, we modify the saddle point problem as follows

$$\begin{aligned} &\min_{v \in \Re^{|\mathcal{S}|}} \max_{\lambda \in \Re^{|\mathcal{S} \times \mathcal{A}|}} \left\{ L(v, \lambda) = \xi^T v + \sum_{a \in \mathcal{A}} \lambda_a^T \left((\gamma P_a - \mathbf{I}) v + r_a \right) \right\}, \\ &\text{subject to} \qquad v \in \mathcal{V}, \qquad \lambda \in \mathcal{U}, \end{aligned} \tag{4}$$

where $\mathcal{V}$ and $\mathcal{U}$ are compact sets that are crafted to take advantages of properties of the Bellman equation. We can show that v^* and λ^* belong to $\mathcal{V}$ and $\mathcal{U}$. As a result, the modified saddle point problem (4) is equivalent to the original problem (3). We can also show that the optimal solution λ^* to the dual problem (4) corresponds to the optimal policy π^* of the MDP. So we only need to find the optimal dual variables to recover the optimal policy.

2.3 The Primal-Dual Algorithm

We are interested in developing algorithms that can find the optimal policies. In particular, we focus on the *model-free learning setting*, which is summarized as below.

Model-Free Learning Setting of MDP

(i) The state space $\mathcal{S}$, the action spaces $\mathcal{A}$, the reward upperbound σ, and the discount factor γ (or the horizon H) are known.

(ii) The transition probabilities $\mathcal{P}$ and reward functions r are unknown.

(iii) There is a Sampling Oracle ($\mathcal{SO}$) that takes input (i, a) and generates a new state j with probabilities $P_a(i, j)$ and a random reward $\hat{r}_{ija} \in [0, \sigma]$ with expectation r_{ija}.

(iv) The space of the algorithm is sublinear to the space of storing the whole transition probabilities.

We provide a high-level outline of the new algorithms as follows. Details of the algorithms can be found in (Chen and Wang, 2016; Wang, 2017).

Algorithm 1 Stochastic Primal-Dual Algorithm for Discounted MDP

```
Input: Sampling Oracle $\mathcal{SO}$, $n = |\mathcal{S}|$, $m = |\mathcal{A}|$, $\gamma \in (0,1)$, $\sigma \in (0,\infty)$
for $k = 1, 2, \ldots, T$ do
    Query $\mathcal{SO}$ for one state-to-state transition and sample reward.
    Formulate the stochastic gradient $(\tilde{g}_v, \tilde{g}_\lambda)$ of the saddle point problem (4) at $(v^{(k-1)}, \lambda^{(k-1)})$
    Update the iterates by
        $v^{(k)} = \text{argmin}_{x \in \mathcal{V}} \left\{ < \eta_v^{(k)} \tilde{g}_v, x > + \Phi_v(x, v^{(k-1)}) \right\}$
        $\lambda^{(k)} = \text{argmin}_{y \in \mathcal{U}} \left\{ < \eta_\lambda^{(k)} \tilde{g}_\lambda, y > + \Phi_\lambda(y, \lambda^{(k-1)}) \right\}$
end for
```

In Algorithm 1, $\eta_v^{(k)}$ is the stepsize of the v-update at k-th iteration and $\eta_\lambda^{(k)}$ is the stepsize of the λ-update. The functions Φ_v and Φ_λ are the two Bregman divergence functions with regard to v and λ respectively. We take Φ_v to be the squared Euclidean distance and we consider two choices for Φ_λ: the squared Euclidean distance and some modified Kullback leiber divergence function (the latter turns out to work better). The algorithm updates the value and policies estimates by processing one sample state-transition at a time. The algorithm keeps track of the dual variable λ (policy estimate) and primal variable v (value estimate), which utilizes $|\mathcal{A}| \times |\mathcal{S}| + |\mathcal{S}| = \mathcal{O}(|\mathcal{A}| \times |\mathcal{S}|)$ space. In comparison, the dimension of the discounted MDP is $|\mathcal{A}|^2 \times |\mathcal{S}|$. Thus the space complexity of Algorithm 1 is sublinear with respect to the model size.

3 Main Results

Let us summarize the main contributions and theoretical results of (Chen and Wang, 2016; Wang, 2017):

1. We study the basic linear algebra of MDPs. We show that the optimal value and optimal policy are dual to each other, and they are the solutions to a stochastic saddle point problem. The *value-policy duality* implies convenient algebraic structure that may facilitate efficient learning and dimension reduction.
2. We develop a class of stochastic primal-dual (SPD) methods which maintain a value estimate and a policy estimate and update their coordinates while processing state-transition data incrementally. The SPD methods exhibit superior space and computational scalability. They require $\mathcal{O}(|\mathcal{S}| \times |\mathcal{A}|)$ space for discounted MDP and $\mathcal{O}(|\mathcal{S}| \times |\mathcal{A}| \times H)$ space for finite-horizon MDP. The space complexity of SPD is sublinear with respect to the input size of the MDP model. For discounted MDP, each iteration updates two coordinates of the value estimate and a single coordinate of the policy estimate. For finite-horizon MDP, each iteration updates $2H$ coordinates of the value estimate and H coordinates of the policy estimate.
3. For discounted MDP, the basic version of SPD (with Φ_v, Φ_λ being squared Euclidean distance) yields an absolute-ϵ-optimal policy with probability at least $1-\delta$ using the following sample size/iteration number
$$\mathcal{O}\left(\frac{|\mathcal{S}||\mathcal{A}|}{(1-\gamma)^4\epsilon^2}\ln\left(\frac{1}{\delta}\right)\right),$$
where $\gamma \in (0,1)$ is the discount factor, $|\mathcal{S}|$ and $|\mathcal{A}|$ are the sizes of the state space and the action space, σ is a uniform upperbound of state-transition rewards. We obtain the sample complexity results by analyzing the duality gap sequence and applying the Bernstein inequality to a specially constructed martingale.
4. For finite-horizon MDP, the basic SPD algorithm yields an absolute-ϵ-optimal policy with probability at least $1-\delta$ using the following sample size/iteration number
$$\mathcal{O}\left(\frac{|\mathcal{S}||\mathcal{A}|H^4}{\epsilon^2}\ln\left(\frac{1}{\delta}\right)\right),$$
where H is the total number of periods. The key aspect of the finite-horizon algorithm is to adapt the learning rate/stepsize for updates on different periods. In particular, the algorithm has to update the policies associated with the earlier periods more aggressively than update those associated with the later periods.
5. We have also proposed an improved version of SPD that takes Φ_v to be some modified Kullback Leiber divergence. For discounted MDP, the improved SPD algorithm yields an ϵ-optimal policy with probability at least $2/3$ using the following sample size/iteration number
$$\mathcal{O}\left(\frac{|\mathcal{S}|^3|\mathcal{A}|\log(|\mathcal{S}||\mathcal{A}|)}{(1-\gamma)^4\epsilon^2}\right),$$
where $\gamma \in (0,1)$ is the discount factor, $|\mathcal{S}|$ and $|\mathcal{A}|$ are the sizes of the state space and the action space. When the Markov decision process is "sufficiently ergodic" for all policies, we show that the complexity of SPD improves to
$$\mathcal{O}\left(\frac{|\mathcal{S}||\mathcal{A}|\log(|\mathcal{S}||\mathcal{A}|)}{(1-\gamma)^2\epsilon^2}\right).$$

The SPD is a model-free method and applies to a wide class of dynamic programming problems. Within the scope of this paper, the sample transitions are drawn from a static distribution. We conjecture that the sample complexity results can be improved by allowing exploitation, i.e., adaptive sampling of actions. The results of this paper suggest that the linear duality of MDP bears convenient structures yet to be fully exploited. We believe that this line of analysis might motivate better methods in the future.

References

D. P. Bertsekas, Dynamic Programming and Optimal Control, vol. I-II, Athena Scientific Belmont, MA, 4th edn., 2007.

D. P. Bertsekas, J. N. Tsitsiklis, Neuro-dynamic programming: an overview, in: Decision and Control, 1995., Proceedings of the 34th IEEE Conference on, vol. 1, IEEE, 560–564, 1995.

R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 1998.

M. L. Puterman, Markov decision processes: discrete stochastic dynamic programming, John Wiley & Sons, 2014.

P. Ortner, R. Auer, Logarithmic online regret bounds for undiscounted reinforcement learning, Advances in Neural Information Processing Systems 19 (2007) 49.

C. Dann, E. Brunskill, Sample complexity of episodic fixed-horizon reinforcement learning, in: Advances in Neural Information Processing Systems, 2818–2826, 2015.

T. Lattimore, M. Hutter, P. Sunehag, Others, The Sample-Complexity of General Reinforcement Learning., in: ICML (3), 28–36, 2013.

T. Lattimore, M. Hutter, PAC bounds for discounted MDPs, in: International Conference on Algorithmic Learning Theory, Springer, 320–334, 2012.

M. Wang, Y. Chen, An online primal-dual method for discounted Markov decision processes, in: 2016 IEEE 55th Conference on Decision and Control (CDC), 4516–4521, 2016.

Y. Chen, M. Wang, Stochastic Primal-Dual Methods and Sample Complexity of Reinforcement Learning, arXiv preprint arXiv:1612.02516 .

M. Wang, Randomized Linear Programming Solves the Discounted Markov Decision Problem In Nearly-Linear Running Time, arXiv preprint arXiv:1704.01869 .

M. G. Azar, R. Munos, M. Ghavamzadeh, H. J. Kappen, et al., Speedy Q-Learning, in: NIPS, 2411–2419, 2011.

M. G. Azar, R. Munos, B. Kappen, On the sample complexity of reinforcement learning with a generative model, arXiv preprint arXiv:1206.6461 .

D. P. de Farias, B. Van Roy, The linear programming approach to approximate dynamic programming, Operations Research 51 (6) (2003) 850–865.

Manipulating Model-based and Model-free Reinforcement Learning in Humans

Maria K Eckstein
Department of Psychology
University of California, Berkeley
Berkeley, CA 94720
`maria.eckstein@berkeley.edu`

Klaus Wunderlich
Department of Psychology
Ludwig Maximilian University, Munich
Geschwister-Scholl-Platz 1, 80539 Munich
`klaus.wunderlich@lmu.de`

Anne GE Collins
Department of Psychology
University of California, Berkeley
Berkeley, CA 94720
`annecollins@berkeley.edu`

Abstract

When deciding what to do, humans and animals employ (at least) two different decision systems: oftentimes, we rely on habits, fixed stimulus-response associations, which have been shaped by past rewards, are computationally cheap, and enable fast responses (”model-free” decision making). But we can also—effortfully—make decisions using planning, mental simulation of different courses of actions and their outcomes, and selection of the one course that leads to the desired goal (”model-based” decision making).

Previous research in humans has shown that it is possible to experimentally reduce model-based decision making relative to model-free decision making, for example by inducing stress (Otto et al., 2013). In the current study, we investigated whether it is also possible to increase model-based decision making. To do this, we implemented a cognitive intervention, which engaged participants in forward-planning and mental simulation (model-based condition), habitual, reward-based processes (model-free condition), or unrelated processes (active control). We assessed decision strategies using the 2-step task (Daw et al., 2011), and fitted a hybrid model-free/model-based reinforcement learning model to estimate participants’ relative weight on each process. In accordance with our pre-registered predictions, we found that the model-based intervention increased the relative weight of model-based versus model-free decision strategies, whereas the model-free intervention had no effect.

These results could have important practical benefits not only for vulnerable populations with known difficulties in decision making, but also for healthy persons who fall back to model-free habit under stress or time pressure, with negative consequences. Nevertheless, more research is needed to investigate the unexpected effect of the control intervention on decision making.

Keywords: Reinforcement Learning Model-based Model-free Decision Making

Acknowledgements

This research was supported by Berkeley’s XLab, which provided the testing space and awarded a grant covering participant payment. The research was also supported a graduate-student fellowship from the German Academic Exchange Service (DAAD).

1 Introduction

Humans make many decisions habitually: for example, we effortlessly navigate the route we take every day, following a fixed sequence of actions, and using landmarks to trigger subsequent actions. But humans also make decisions in a goal-directed way. For example, when we plan how to reach a new destination, we flexibly combine individual pieces into a new route, using a cognitive map or model of our environment. These two different modes of decision making, habitual and goal-directed, have long been differentiated in psychology, and form the basis of two different schools of thought, namely Behaviorism (e.g., Skinner, 1977) and cognitivism (e.g., Tolman, 1948). A parallel differentiation between decision making strategies exists in reinforcement learning, a branch of machine learning. Here, the distinction is between model-based (MB; similar to goal-directed) and model-free (MF; similar to habitual) agents. MB agents use a model of the environment to simulate possible actions and outcomes, and then determine which actions are expected to lead to the best outcomes. In contrast, MF agents determine the value of actions by accumulating the past reward history of these actions.

Ever more often, reinforcement learning algorithms are applied in psychological research. This has led to the discovery that activity in the brain's dopaminergic "reward system" (Wise and Rompre, 1989) coincides with the occurrence of reward prediction errors as specified in MF reinforcement learning (Schultz, Dayan, and Montague, 1997). MB learning, on the other hand, has been shown to rely on a distinct brain network including frontal cortex and dorsal striatum (Dolan and Dayan, 2013). Human learning and decision making relies on both MB and MF processes (Daw, Gershman, Seymour, Dayan, and Dolan, 2011), and a key question is how we arbitrate between the two. Previous studies have shown that cognitively demanding MB decision making is less prevalent when time or cognitive resources are sparse, for example during stress (Schwabe and Wolf, 2011) or multi-tasking (Otto, Gershman, Markman, and Daw, 2013). On the other hand, no situations have yet been identified that increase MB decision making. The only study that has shown an increase was a pharmacological manipulation of dopamine levels (Wunderlich, Smittenaar, and Dolan, 2012).

In the current study, we therefore sought to investigate whether a cognitive intervention could increase MB decision making. Cognitive strategies are influenced by prior cognitive activities (Jaeggi, Buschkuehl, Jonides, and Shah, 2011; Muraven and Baumeister, 2000). Thus, we predicted that we could affect MB decision processes by training participants on tasks involving forward-planning and mental simulation, whereas training on tasks involving habitual, reward-driven behavior should affect the MF process. Here, we test this prediction in a behavioral study. The study was pre-registered on the Open Science Framework prior to data collection (osf.io/nw9vz).

2 Methods and Results

2.1 Study Design and Description of the Tasks

116 participants took part in the two-session experiment. In session 1, all participants first performed the 2-step decision making task (Daw et al., 2011, see description below; Figure 1B), then received one of three training interventions designed at testing our hypothesis, then were tested on the 2-step task again (run 2). Participants came back for a third assessment of the 2-step task 2 days later (run 3; Figure 1A).

Participants were randomly assigned to one of three interventions: model-based (MB), model-free (MF), or control. We chose tasks that were well established in the literature for engaging cognitive and neural processes corresponding to each mode of decision making (MB and MF), or for not engaging MB or MF processes (control). For training, MB participants engaged in two tasks that were closely related to MB decision making: a planning-intensive version of the Tower of London task (Beauchamp, Dagher, Aston, and Doyon, 2003), and a rule-based category learning task (Maddox and Ashby, 2004). Both tasks engage model-based planning or cognitive control, and rely on the brain network required for MB decision making (Dolan and Dayan, 2013). MF participants engaged in tasks targeted at MF processes: a habitual reward-based task (Tricomi and Lempert, 2015) and an information-integration category learning task (Maddox and Ashby, 2004). Both tasks engage long-term information integration and habitual behaviors, and rely on brain regions underlying MF decisions (Dolan and Dayan, 2013). Finally, the control tasks (number comparison task: Piazza, 2010; orientation discrimination task: Sasaki, Nanez, and Watanabe, 2010) were unrelated to MB or MF decision making.

Participants' decision strategies were assessed using the 2-step task (Daw et al., 2011; see Figure 1B). On each trial, this task involves two sequential binary decisions, which are potentially followed by a reward. The first decision determines with high probability what the second set of choices will be; however, in rare transition cases, the opposite set of choices is offered instead. This task is designed so that MB and MF strategies are distinguishable. A MF agent learns the values of a state-one action a_1 by accumulating the rewards obtained in trials in which a_1 was selected; future choices of a_1 are therefore mainly driven by past reward, independently of whether the trial included a common or rare transition between s_1 and s_2. A MB agent, on the other hand, selects a_1 based on its knowledge of the task structure, taking into account the transition probabilities between state s_1 and s_2 when reasoning which action a_1 to take. Future actions a_1 therefore depend on both past rewards and transition probabilities. Specifically, MB agents tend to repeat a_1 upon reward

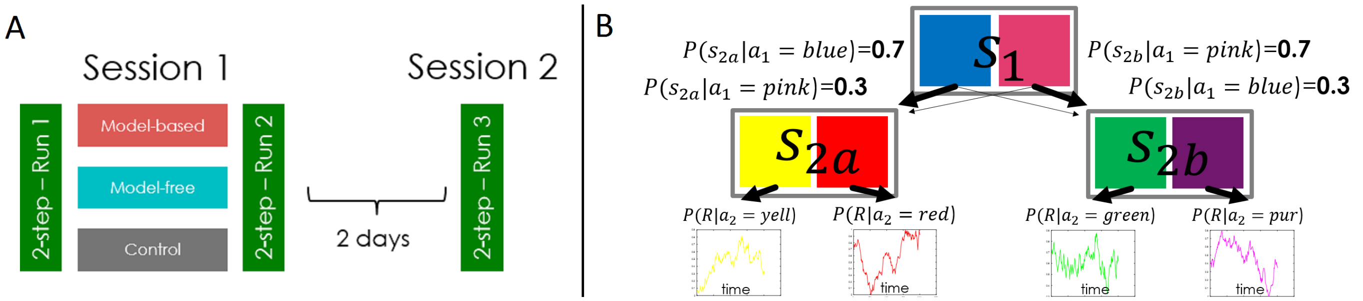


Figure 1: Experimental procedure (A) and 2-step task (B). (A) as described in the main text. (B) 2-step task: each trial has two successive states, s_1 and s_2. In s_1, participants decide between two actions a_1 (blue vs pink). One choice (e.g. blue) is followed by state s_{2a} most of the time ("common transition"), the other one (pink) by state s_{2b}. After deciding between two actions in state s_2, participants receive a reward with probability $P(R|a_2)$, which only depends on a_2. Reward probabilities change over time to ensure continued learning.

when the transition was common, but select the alternative a_1 upon reward when the transition was rare. The opposite pattern emerges in unrewarded trials.

2.2 Behavioral Switch-Stay Analysis

We tested 116 participants in this paradigm. Standard procedures were used for data cleaning and exclusion of participants, resulting in 303 datasets from 114 participants (110 in run 1: 44 MB, 43 MF, 21 control; 103 in run 2: 43 MB, 41 MF, 19 control; and 90 in run 3: 36 MB, 35 MF, 19 control). We first analyzed the 2-step task using logistic regression to assess the effect of rewards and transitions on future choices (Akam, Costa, and Dayan, 2015; see Figure 2A and B). Participants repeated rewarded actions more often than unrewarded actions, a sign of MF behavior. This effect was statistically significant in all runs in the MF and MB groups, but not in the control group, as revealed by main effects of previous reward on staying, in logistic mixed-effects regression models controlling for choice repetition and key repetition (control group: all $\beta's < 0.12$, $z's < 1.61$, $p's > 0.11$; MB: all $\beta's > 0.15$, $z's > 3.79$, $p's < .001$; MF: all $\beta's > 0.18$, $z's > 3.41$, $p's < .001$). Besides this MF component, participants also showed markers of MB decision making, revealed by significant positive interactions between reward and transition, in run 2 (control: $\beta = 0.11$, $z = 2.05$, $p = .040$; MB: $\beta = 0.11$, $z = 2.54$, $p = .011$; MF: $\beta = 0.10$, $z = 2.92$, $p = .0035$). We then tested for differences between groups, using interaction contrasts. We found no differences in the MF or MB component for any run, shown by non-significant interactions between reward and group, all $\chi^2(2) < 1.33$, $p's > .40$, and between reward, transition, and group, all $\chi^2(2) < 1.17$, $p's > .56$. Lastly, we found that the model-based component changed over time, as revealed by the interaction between reward, transition, and run, $\chi^2(4) < 15.07$, $p = .0046$. However, this change over time did not differ by group, as revealed by the non-significant interaction of this effect with training, $\chi^2(16) < 6.04$, $p = .99$). Thus, it probably reflected practice, rather than intervention effects.

In summary, the MB and MF groups showed MF characteristics in all runs and additional MB components in run 2. The control group showed no sign of MF decision making, but MB decisions in run 2. These results were confirmed by regression models integrating a larger number of previous trials to predict actions (Akam et al., 2015; results not shown).

2.3 RL Modeling Analysis

The previous analyses focus on the influence of a single previous trial on decision making. We followed these analyses up with computational modeling to assess participants' decision strategies as a mixture of long-term MB vs. MF reinforcement learning (Akam et al., 2015; Daw et al., 2011; Sutton and Barto, 2017). We specified a hybrid model, in which agents determine action values by combining MB and MF value estimates. We then fit this model to each participant's actions by selecting parameter values that maximized the likelihood of the observed actions under the model. The parameter w, which determines the weight of MB and MF value estimates, was used to assess participants' decision strategies.

Our model was similar to previously published models of this task (e.g., Wunderlich et al., 2012). Specifically, agents update action values Q for actions a_2 in the second state s_2 by observing the trial outcome (reward $R = 1$ or $R = 0$):

$$Q(s_2, a_2) = Q(s_2, a_2) + \alpha_2 \cdot RPE, \tag{1}$$

where the reward prediction error $RPE = R - Q(s_2, a_2)$ and α_2 is the agent's learning rate in the second state. The update of first-state action values $Q(s_1, a_1)$ differs between MB and MF agents. MF agents use the outcome of a_1 to update $Q(s_1, a_1)$. The outcome of a_1 consists in the value of the action chosen in s_2 and the trial's reward R.

$$Q_{mf}(s_1, a_1) = Q_{mf}(s_1, a_1) + \alpha_1 \cdot (VPE + \lambda \cdot RPE), \tag{2}$$

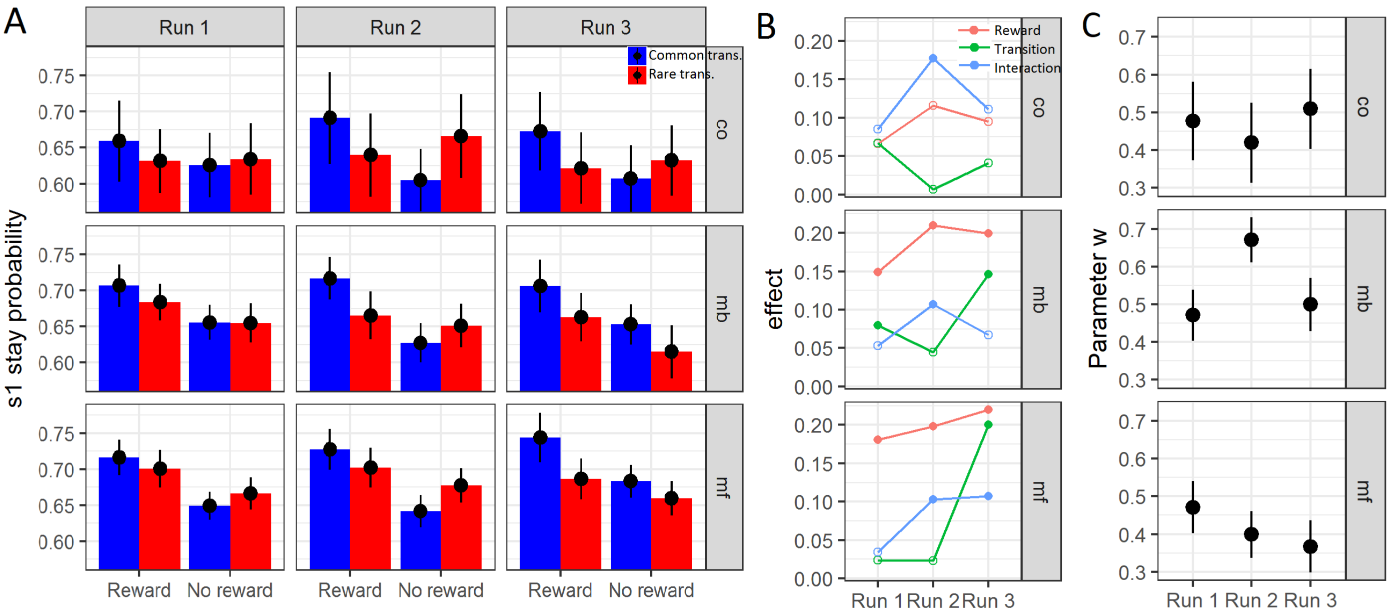


Figure 2: Results of the 2-step analyses. (A) Participants' probability of repeating the action a_1 taken in the previous trial ("s1 stay probability"), as a function of reward and transition in the previous trial, including standard errors (SE). (B) Beta weights of logistic regression predicting staying (repeating previous a_1) from previous reward, transition, and their interaction. Empty circles: $p >= 0.05$; filled circles: $p < 0.05$. (C) Means and SE of fitted parameter w.

where the value prediction error $VPE = Q_{mf}(s_2, a_2) - Q_{mf}(s_1, a_1)$. The weight of the RPE is determined by λ, a temporal discounting factor.

MB agents determine $Q(s_1, a_1)$ based on an internal predictive model, taking into account the transition probability $p(s_2, a_1, s_1)$ between states s_1 and s_2 upon choosing a_1, and planning to select the best available action a_2 thereafter:

$$Q_{mb}(s_1, a_1) = \sum_{s_2} p(s_2, a_1, s_1) \cdot max(Q(s_2, a_2)). \tag{3}$$

Agents combine MB and MF value estimates using a weighted average, $Q_{hyb}(s_1, a_1) = (1 - w) \cdot Q_{mf}(s_1, a_1) + w \cdot Q_{mb}(s_1, a_1)$. The parameter w determines the weight of MB versus MF values. Agents select actions a according to a softmax decision rule, which takes into account the action's value $Q(s, a)$, but also whether the same action was taken in the previous trial (choice perseverance p) and whether the same key was used to select it (key perseverance k). The inclusion of k is an extension of previous models and improved the model fit significantly. We validated our model by simulating agents with different parameter values and subsequently recovering these parameters (data not shown).

We then aimed to characterize human performance in terms of the model parameters, specifically parameter w indicating the balance between MB and MF decision making. Model comparison with Bayes Information Criterion (BIC) indicated that a full hybrid model albeit with fixed future discounting $\lambda = 1$ was best (Wilcoxon signed-ranks test, $w = 61964$, $p < .001$). In accordance with the previous analyses, model fitting results (Figure 2B) showed that w increased from run 1 (before intervention) to run 2 (right after intervention) in the MB group, as revealed by a repeated-measures t-test, $t(36) = 2.25$, $p = 0.031$. This was not the case in the MF or control group (MF: $t(36) = 0.80$, $p = 0.43$; control: $t(15) = 0.46$, $p = 0.65$). Nevertheless, no long-term effects of the training were evident (all $ts < 0.24$, all $ps > .81$). A mixed-effects regression model supported these results. We tested for effects of group and run on w (group, contrast of MB versus MF and control: $\beta = 0.036$, $p = .039$; interaction contrast with run: $\beta = 0.063$, $p = .033$).

3 Conclusion

In this study, we aimed at investigating which factors might influence human decision making. Specifically, we tested whether the use of MB versus MF strategies depends on previous cognitive activities, such that forward planning and rule-based category learning would increase MB decision strategies, whereas habitual, reward-driven stimulus-response behavior would increase MF decision making. In accordance with our hypotheses, we found that decision strategies became more model-based after the MB intervention involving forward planning and rule-based category learning, whereas the MF and control interventions had no effect.

These results support our hypothesis that model-based decision making is closely related to forward-planning and rule-based category learning. Indeed, forward-planning and model-based decision making both rely on the employment of cognitive models. We have shown that creating and employing cognitive models in one task facilitates subsequent model-based reasoning in a completely unrelated task. In other words, humans seem to be able to learn model use on a highly abstract level. Rather than relying more on one specific model or one kind of models, participants learned to rely more on cognitive models in general.

Transitioning to rule-based category learning, previous research has already suggested similarities to model-based reasoning, both with respect to underlying cognitive processes and brain areas (Maddox & Ashby, 2004). Nevertheless, this assumption had not yet been tested experimentally. Our findings support the claim that both kinds of cognitive activities indeed engage similar processes, despite their superficial differences. Future research will be needed to assess the exact overlap.

Another open question for future research is how the effects of forward-planning and rule-based category learning combined in producing the observed increase in model-based decision making. It is possible that the effects were driven by just one of the training tasks or that both interacted to produce the results. Having established that it is possible to increase model-based decision making with a combination of both tasks, future studies will be needed to disentangle the contributions of either one.

Summing up, research into the manipulation of MB and MF decision strategies is of great relevance. MF decisions are fast and efficient, but can be suboptimal in the long term (eating the chocolate cake). MB decisions (preparing for an exam) require more effort but can often lead to better long-term outcomes. Many people, including persons with clinical conditions such as ADHD, depression, or eating disorders, would benefit from a cognitive intervention that facilitates MB decision making and trains selecting appropriate decision strategies. Further research is needed to establish the usefulness of interventions similar to the one presented here in clinical populations.

4 References

References

Akam, T., Costa, R., & Dayan, P. (2015). Simple Plans or Sophisticated Habits? State, Transition and Learning Interactions in the Two-Step Task. *PLoS Comput Biol*, *11*(12), e1004648.

Beauchamp, M. H., Dagher, A., Aston, J. a. D., & Doyon, J. (2003). Dynamic functional changes associated with cognitive skill learning of an adapted version of the Tower of London task. *NeuroImage*, *20*(3), 1649–1660.

Daw, N. D., Gershman, S. J., Seymour, B., Dayan, P., & Dolan, R. J. (2011). Model-Based Influences on Humans' Choices and Striatal Prediction Errors. *Neuron*, *69*(6), 1204–1215. doi:10.1016/j.neuron.2011.02.027

Dolan, R. J. & Dayan, P. (2013). Goals and Habits in the Brain. *Neuron*, *80*(2), 312–325. doi:10.1016/j.neuron.2013.09.007

Jaeggi, S. M., Buschkuehl, M., Jonides, J., & Shah, P. (2011). Short- and long-term benefits of cognitive training. *Proceedings of the National Academy of Sciences*, *108*(25), 10081–10086. doi:10.1073/pnas.1103228108

Maddox, W. T. & Ashby, F. G. (2004). Dissociating explicit and procedural-learning based systems of perceptual category learning. *Behavioural Processes*, *66*(3), 309–332.

Muraven, M. & Baumeister, R. F. (2000). Self-regulation and depletion of limited resources: Does self-control resemble a muscle? *Psychological bulletin*, *126*(2), 247.

Otto, A. R., Gershman, S. J., Markman, A. B., & Daw, N. D. (2013). The curse of planning dissecting multiple reinforcement-learning systems by taxing the central executive. *Psychological science*, 0956797612463080.

Piazza, M. (2010). Neurocognitive start-up tools for symbolic number representations. *Trends in Cognitive Sciences*, *14*(12), 542–551. doi:10.1016/j.tics.2010.09.008

Sasaki, Y., Nanez, J. E., & Watanabe, T. (2010). Advances in visual perceptual learning and plasticity. *Nature reviews. Neuroscience*, *11*(1), 53–60. doi:10.1038/nrn2737

Schultz, W., Dayan, P., & Montague, P. R. (1997). A Neural Substrate of Prediction and Reward. *Science*, *275*(5306), 1593–1599. doi:10.1126/science.275.5306.1593

Schwabe, L. & Wolf, O. T. (2011). Stress-induced modulation of instrumental behavior: From goal-directed to habitual control of action. *Behavioural brain research*, *219*(2), 321–328.

Skinner, B. F. (1977). Why I am not a cognitive psychologist. *Behaviorism*, 1–10.

Sutton, R. S. & Barto, A. G. (2017). *Reinforcement Learning: An Introduction* (2nd ed.). Cambridge, MA; London, England: MIT Press.

Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological review*, *55*(4), 189.

Tricomi, E. & Lempert, K. M. (2015). Value and probability coding in a feedback-based learning task utilizing food rewards. *Journal of Neurophysiology*, *113*(1), 4–13. doi:10.1152/jn.00086.2014

Wise, R. A. & Rompre, P.-P. (1989). Brain dopamine and reward. *Annual review of psychology*, *40*(1), 191–225.

Wunderlich, K., Smittenaar, P., & Dolan, R. J. (2012). Dopamine Enhances Model-Based over Model-Free Choice Behavior. *Neuron*, *75*(3), 418–424. doi:10.1016/j.neuron.2012.03.042

Policy Iteration for Discounted Reinforcement Learning Problems in Continuous Time and Space

Jae Young Lee[*]

Reinforcement Learning and Artificial Intelligence
Department of Computing Science
University of Alberta,
4-08 Athabasca Hall, Edmonton, AB, Canada, T6G 2E8
`jyounglee@ualberta.ca`

Richard S. Sutton

Reinforcement Learning and Artificial Intelligence
Department of Computing Science
University of Alberta,
2-21 Athabasca Hall, Edmonton, AB, Canada T6G 2E8
`rsutton@ualberta.ca`

Abstract

Recent advances in various fields regarding decision making, especially regarding reinforcement learning (RL), have revealed the interdisciplinary connections among their findings. For example, actor and critic in computational RL are shown to play the same roles of dorsal and ventral striatum; goal-directed and habitual learning is strongly relevant to model-based and model-free computational RL, respectively. Among the different methodologies in those fields, theoretical approach in machine learning community has established the well-defined computational RL framework in discrete domain and a dynamic programming method known as *policy iteration* (PI), both of which served as the fundamentals in computational RL methods. The main focus of this work is to develop such RL framework and a series of PI methods in continuous domain, with its environment modeled by an ordinary differential equation (ODE). Similar to the discrete case, the PI methods are designed to recursively find the best decision-making strategy by iterating *policy evaluation* (as a role of *critic*) and *policy improvement* (as a role of *actor*). Each proposed one is either model-free corresponding to habitual learning, or partially model-free (or partially model-based) corresponding to somewhere between goal-directed (model-based) and habitual (model-free) learning. This work also provides theoretical background and perhaps, the basic principles to RL algorithms with a real physical task which is usually modeled by ODEs. In detail, we propose on-policy PI and then four off-policy PI methods—the two off-policy methods are the ideal PI forms of advantage updating and Q-learning, and the other two are extensions of the existing off-policy PI methods; compared to PI in optimal control, ours do not require an initial stabilizing policy. The mathematical properties of admissibility, monotone improvement, and convergence are all rigorously proven; simulation examples are provided to support the theory.

Keywords: policy iteration, reinforcement learning, off-policy, discounting, continuous time, continuous space, convergence, model-free, partially model-free, ordinary differential equation, actor-critic, Hamilton-Jacobi-Bellman

Acknowledgements

The authors gratefully acknowledge the support of Alberta Innovates – Technology Futures, the Alberta Machine Intelligence Institute, Google Deepmind, and the Natural Sciences and Engineering Research Council of Canada.

---

[*] Corresponding author.

1 Introduction

Decision making problems have been studied in various disciplines such as machine learning, neuroscience, psychology, and optimal control, with a number of different methodologies, e.g., simulation-based, theoretical, biological, and empirical approaches. Recent advances in those fields regarding decision making, especially regarding reinforcement learning (RL), have revealed the interdisciplinary connections among their findings. Closely related to this work is that the actor-critic structure in computational RL presumably exists in a brain mechanism of dorsal (actor) and ventral (critic) striatum, and that model-based and model-free computational RL are very relevant to goal-directed and habitual learning in psychology, respectively [5].

Among the different approaches, the theoretical works in machine learning community have established the fundamental mathematical model of the computational RL frameworks, with its environment represented by a finite Markov decision process (MDP), and a dynamic programming method known as *policy iteration* (PI) with its mathematical properties. Here, PI recursively finds the best decision-making strategy, called the optimal policy, with *policy evaluation* as a role of *critic* and *policy improvement* as a role of *actor* [5]. Together with the other dynamic programming methods, PI has served as a fundamental principle to develop computational RL methods in the MDP framework.

On the other hand, different from the MDP environment, the dynamics of real physical world is usually modeled by ordinary differential equations (ODEs) in continuous time and space (CTS). In such continuous domain, a similar PI method has also recently come to be studied in optimal control fields [4]. An interesting point of such a PI method is that it is partially model-free, lying somewhere on the line between goal-directed (model-based) and habitual (model-free) learning in psychology. This is a quite different situation from the MDP case, where PI is completely model-based, but the actor-critic methods developed so far are all model-free and thus thought to be relevant only to habitual behavior [5]. There are also off-policy versions of PI in CTS, each of which is either completely or partially model-free [3]. However, it is not theoretically straightforward to extend such PI methods in CTS from optimal control to the general RL framework.

In this work, we precisely define the RL framework in CTS, with its environment represented by an ODE, and then propose a series of PI methods in that framework—one is the extension of integral PI [3, 4], and the other four are its off-policy versions (two are the ideal PI forms of advantage updating [1, 2] and Q-learning in CTS, and the other two correspond to the off-policy PI methods in [3]). Similar to PI in the MDP environment, their mathematical properties of admissibility, monotone improvement, and convergence to the optimal solution are all rigorously proven. As opposed to the PI methods in optimal control, all of the proposed PI schemes do not require an initial stabilizing policy, by virtue of discounting, while each one still remains to be either completely or partially model-free. Simulation examples are also provided to support the theory. Though our PI methods are not online incremental RL algorithms, we believe that this work provides the theoretical background of and intuition to the RL methods in CTS, e.g., those in [1, 2]. This theoretical work, lying between the fields of machine learning and optimal control, would also provide some motivational links in the future between the RL methods and the findings in neuroscience and psychology in CTS, perhaps similar to the interdisciplinary links between them in discrete time and space. For brevity, all of the Theorem proofs are omitted.

2 RL Problem Formulation in CTS

In the RL problem, $\mathcal{X} \doteq \mathbb{R}^n$ denotes the state space, and the action space $\mathcal{U} \subseteq \mathbb{R}^m$ is a m-dimensional manifold in $\mathbb{R}^m$ with (or without) boundary; $t \geq 0$ denotes a given specific time instant; the environment in CTS is described by an ODE $\dot{X}_\tau = f(X_\tau, U_\tau)$, where $\tau \in [t, \infty)$ is the time variable, $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is a continuous function, $X_\tau \in \mathcal{X}$ is the state vector at time τ with its time-derivative $\dot{X}_\tau \in \mathcal{X}$, and the action trajectory $U_\tau \in \mathcal{U}$ is a right continuous function over $[t, \infty)$. A continuous function $\pi : \mathcal{X} \to \mathcal{U}$ is called a (stationary) policy whenever the state trajectory $\mathbb{E}_\pi[X_\tau | X_t = x]$ is uniquely defined for all $\tau \geq t$ and all $x \in \mathcal{X}$, where $\mathbb{E}_\pi[Z | X_t = x]$ has no stochastic role but means the deterministic value Z when $X_t = x$ and $U_\tau = \pi(X_\tau)$ for all $\tau \geq t$. We will denote $\Delta t > 0$ the time difference, $t' \doteq t + \Delta t$, and $X_t' \doteq X_{t'}$. The RL problem we consider is to find the optimal policy π_* that maximizes the value function (VF) $v_\pi : \mathcal{X} \to \mathbb{R} \cup \{-\infty\}$

$$v_\pi(x) \doteq \mathbb{E}_\pi[G_t | X_t = x] \text{ with a discounted return } G_t \doteq \int_t^\infty \gamma^{\tau - t} R_\tau \, d\tau \ \text{ (upper bounded)}, \tag{1}$$

where $R_\tau \doteq R(X_\tau, U_\tau) \in \mathbb{R}$ is the immediate reward at time τ and $\gamma \in (0, 1)$ is the discount factor. The reward function $R : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ here is continuous and upper-bounded. A policy π (or its VF v_π) is said to be admissible, denoted by $\pi \in \Pi_a$ or $v_\pi \in \mathcal{V}_a$, if $v_\pi(x)$ is finite for all $x \in \mathcal{X}$, where Π_a and $\mathcal{V}_a$ denote the set of all admissible policies and VFs, respectively. Note that if R is bounded, then every possible policy π is guaranteed to be admissible and has a bounded value function. In our RL problem, we assume that every $v_\pi \in \mathcal{V}_a$ has its continuous gradient ∇v_π and that there is an optimal *admissible* policy π_* such that $v_\pi(x) \leq v_*(x)$ for any $x \in \mathcal{X}$ and any policy π, where v_* is the optimal VF.

3 Partially Model-Free Policy Iteration

Note that in our CTS case, any $v_\pi \in \mathcal{V}_a$ satisfies

$$\forall x \in \mathcal{X} \text{ and } \forall \Delta t > 0 : \begin{cases} \text{the Bellman equation: } v_\pi(x) = \mathbb{E}_\pi\Big[\mathcal{R}_t + \gamma^{\Delta t}\, v_\pi(X_t')\Big|X_t = x\Big] \text{ with } \mathcal{R}_t \doteq \int_t^{t'} \gamma^{\tau - t}\, R_\tau \, d\tau; \\ \text{the boundary condition: } \lim_{k\to\infty} \gamma^{k\Delta t}\, \mathbb{E}_\pi[v_\pi(X_{t+k\Delta t})|X_t = x] = 0. \end{cases}$$

The policy improvement operation in CTS is defined in the limit $\Delta t \to 0$ as

$$\begin{aligned} \pi'(x) &\in \operatorname*{arg\,max}_{U_t \in \mathcal{U}} \lim_{\Delta t \to 0} \frac{1}{\Delta t} \cdot \mathbb{E}\Big[\mathcal{R}_t + \gamma^{\Delta t}\, v_\pi(X_t') - v_\pi(X_t)\Big|X_t = x\Big] \\ &= \operatorname*{arg\,max}_{u \in \mathcal{U}} \big(R(x,u) + \dot{v}_\pi(x,u) + \ln\gamma \cdot v_\pi(x)\big) \quad \forall x \in \mathcal{X}, \end{aligned} \tag{2}$$

where π' is the improved policy, and $\dot{v}_\pi(x,u) = \nabla v_\pi(x) f(x,u)$ by chain rule.

Considering any decomposition: $f(x,u) = f_d(x) + f_c(x,u)$, where f_d is an *unknown* drift dynamics and f_c is a known input coupling dynamcs, and noting that any addition or subtraction of u-independent terms does not change the maximization process *with respect to* $u \in \mathcal{U}$, one can express (2) in terms of f_c as

$$\pi'(x) \doteq \operatorname*{arg\,max}_{u \in \mathcal{U}} \Big(R(x,u) + \nabla v_\pi(x) f_c(x,u)\Big) \tag{3}$$

which we call partially model-free policy improvement. Algorithm 1 is our partially model-free PI which finds the optimal solution without knowing the drift dynamics f_d. It starts with an initial admissible policy π_0; in each i-th step, it finds a function v_i satisfying the Bellman equation (policy evaluation), and then using v_i and f_c, the next policy π_{i+1} is updated by "(3) with $\pi' = \pi_{i+1}$ and $v_\pi = v_i$" (policy improvement). This process is recursively done until convergence. Our main theorem is as follows.

Algorithm 1: Partially Model-Free Policy Iteration

```
Initialize: $i \leftarrow 0$, $\Delta t > 0$, and $\pi_0 \in \Pi_a$;
repeat
    Policy Evaluation: find $v_i : \mathcal{X} \to \mathbb{R}$ such that $\forall x \in \mathcal{X}$:
        $v_i(x) = \mathbb{E}_{\pi_i}\Big[\mathcal{R}_t + \gamma^{\Delta t}\, v_i(X_t')\Big|X_t = x\Big]$;
    Policy Improvement: find a policy $\pi_{i+1}$ such that
        $\pi_{i+1}(x) \in \operatorname*{arg\,max}_{u \in \mathcal{U}} \Big(R(x,u) + \nabla v_i(x) f_c(x,u)\Big) \;\; \forall x \in \mathcal{X}$;
    $i \leftarrow i + 1$;
until convergence is met.
```

Assumption 1. *(Policy improvement condition) for each $\pi \in \Pi_a$, there is a policy π' such that* (3) *holds.*

Assumption 2. *(Boundary condition) $\forall i \in \mathbb{Z}_+$: if π_i is admissible, then* $\lim_{k\to\infty} \gamma^{k\Delta t}\, \mathbb{E}_{\pi_i}[v_i(X_{t+k\Delta t})|X_t = x] = 0 \quad \forall x \in \mathcal{X}$.

Assumption 3. *(Uniqueness of optimality) there is one and only one element $w_* \in \mathcal{V}_a$ over $\mathcal{V}_a$ that satisfies*

$$\textit{the Hamilton-Jacobi-Bellman equation: } 0 = \max_{u \in \mathcal{U}} \big(R(x,u) + \dot{w}_*(x,u) + \ln\gamma \cdot w_*(x)\big) \quad \forall x \in \mathcal{X} \;\; \textit{(in fact, } w_* = v_*\textit{)}.$$

Theorem 1. *The sequences $\{\pi_i\}_{i=0}^\infty$ and $\{v_i\}_{i=0}^\infty$ generated by Algorithm 1 under Assumptions 1–3 satisfy the followings.*

- (P1) $\pi_{i+1} \in \Pi_a$ *and* $v_i = v_{\pi_i} \in \mathcal{V}_a$ *for all* $i \in \mathbb{N} \cup \{0\}$;
- (P2) *the policy is monotonically improved, i.e.,* $v_{\pi_0}(x) \le v_{\pi_1}(x) \le \cdots \le v_{\pi_i}(x) \le v_{\pi_{i+1}}(x) \le \cdots \le v_*(x)$ *for all* $x \in \mathcal{X}$;
- (P3) $v_i \to v_*$ *with respect to some metric* $d : \mathcal{V}_a \times \mathcal{V}_a \to [0, \infty)$, *i.e.,* $\lim_{i\to\infty} d(v_i, v_*) = 0$;
- (P4) $v_i \to v_*$ *pointwisely on $\mathcal{X}$ and uniformly on any compact subset of $\mathcal{X}$ if:*
  - *(a) the limit function* $\hat{v}_* \doteq \lim_{i\to\infty} v_i$ *belongs to* $\mathcal{V}_a$;
  - *(b) for every compact subset $\Omega \subset \mathcal{X}$, the policy iteration mapping $v_\pi \mapsto v_{\pi'}$ is continuous with respect to*

$$\textit{the uniform pseudometric } d_\Omega(v,w) \doteq \sup_{x\in\Omega}\big|v(x) - w(x)\big| \;\; (v, w \in \mathcal{V}_a).$$

4 Extensions to Off-Policy Policy Iteration

PI shown in Algorithm 1 can be extended to a series of its off-policy versions that use the behavior policy μ than the target policy π_i to generate the state trajectory X_τ (and the reward R_τ). To describe μ, we extend the concept of a policy established in Section 2. A function $\mu : [t, \infty) \times \mathcal{X} \to \mathcal{U}$ is called a (non-stationary) policy if: 1) $\mu(\tau, \cdot)$ is continuous for each fixed $\tau \ge t$ and $\mu(\cdot, x)$ is right continuous for each fixed $x \in \mathcal{X}$; 2) for each $x \in \mathcal{X}$, the state trajectory $\mathbb{E}_\mu^x[X_\tau]$ is uniquely defined for all $\tau \ge t$. Here, $\mathbb{E}_\mu^x[Z]$ means the deterministic value Z when $X_t = x$ and $U_\tau = \mu(\tau, X_\tau)\ \forall \tau \ge t$. A function $\mu : [t, \infty) \times \mathcal{X} \times \mathcal{U}_0 \to \mathcal{U}$ is said to be an AD policy over $\mathcal{U}_0 \subseteq \mathcal{U}$ if for each fixed $u \in \mathcal{U}_0$, $\mu(\cdot, \cdot, u)$ is a policy and $\mu(t, x, u) = u$ holds for all $x \in \mathcal{X}$ and all $u \in \mathcal{U}_0$. For an AD policy μ, we denote $\mathbb{E}_{\mu(\cdot,\cdot,u)}^x[Z]$ by $\mathbb{E}_\mu^{(x,u)}[Z]$.

By replacing policy evaluation and improvement, we propose four different off-policy PI methods—Advantage PI (API), QPI, Explorized PI (EPI), and Common PI (CPI). In policy evaluation of API, the advantage function a_π defined as $a_\pi(x,u) \doteq R(x,u) + \dot{v}_\pi(x) + \ln\gamma \cdot v_\pi(x)$ [1, 2] is estimated, along with v_π and the constraint $a_\pi(x,\pi(x)) = 0$. In QPI, the Q-function q_π so-defined in CTS as $q_\pi(x,u) = \kappa \cdot v_\pi(x) + a_\pi(x,u)$ for some $\kappa \neq 0$ is estimated in policy evaluation with the discounting $\beta \doteq \gamma \cdot e^\kappa > 0$ that should be different from $\gamma \in (0,1)$. Here, β determines κ in q_π, and the extremely large $|\kappa|$ may result in a significant performance degradation or extremely slow Q-learning [1]. Both a_π in API and q_π in QPI replace the policy improvement (3) with the respective model-free ones. EPI is the direct extension of PI with respect to the behavior policy μ without introducing any other function than v_π. CPI is the model-free modification of EPI when

- (C1) $f_c(x,u) = F_c(x)u$ for a continuous function $F_c : \mathcal{X} \to \mathbb{R}^{n\times m}$ (input-affine dynamics);
- (C2) $\mathcal{U}$ is convex and the reward R is given by $R(x,u) = R_0(x) - S(u)$ for a continuous upper-bounded function R_0 and a strictly convex function S, with its gradient $\nabla S : \mathcal{U} \to \mathbb{R}^{1\times m}$ that is continuous and has its inverse ∇S^{-1} on the interior of the action space domain $\mathcal{U}$.

The key idea of CPI is to estimate the C-function $c_\pi(x) \doteq F_c^\mathsf{T}(x)\nabla v_\pi^\mathsf{T}(x)$ in policy evaluation and then use it in policy improvement under (C1) and (C2) above. Here, note that the maximization (3) (and thus policy improvement of PI, EPI, and CPI) can be dramatically simplified under (C1) and (C2) as $\pi'(x) = \sigma(F_c^\mathsf{T}(x)\nabla v_\pi^\mathsf{T}(x)) = \sigma(c_\pi(x))$ with $\sigma^\mathsf{T} \doteq \nabla S^{-1}$.

The policy evaluation and improvement of the off-policy methods are summarized in Table 1, where we used the compact notations $R^{\pi_i} \doteq R(\cdot,\pi_i(\cdot))$, $a_i^{\pi_i} \doteq a_i(\cdot,\pi_i(\cdot))$, $q_i^{\pi_i} \doteq q_i(\cdot,\pi_i(\cdot))$, $f_c^{\pi_i} \doteq f_c(\cdot,\pi_i(\cdot))$, $\xi_\tau^{\pi_i} \doteq U_\tau - \pi_i(X_\tau)$, and

$$\mathcal{I}_\alpha(Z) \doteq \int_t^{t'} \alpha^{\tau - t} Z(X_\tau, U_\tau)\, d\tau \quad \text{and} \quad D_\alpha(v) \doteq v(X_t) - \alpha^{\Delta t} v(X_t')$$

for brevity. For example, the policy evaluation of Algorithm 1 can be expressed as $\mathbb{E}_{\pi_i}^x[D_\gamma(v_i)] = \mathbb{E}_{\pi_i}^x[\mathcal{I}_\gamma(R)]$. As shown in Table 1, API, QPI, and CPI are model-free while EPI requires the full-knowledge of an input-coupling dynamics f_c to run. On the other hand, while API and QPI explore the whole state-action space $\mathcal{X}\times\mathcal{U}$ to learn their respective functions (v_π, a_π) and q_π for all $(x,u) \in \mathcal{X}\times\mathcal{U}$, EPI and CPI search only the significantly smaller spaces $\mathcal{X}$ and $\mathcal{X}\times\{u_j\}_{j=0}^m$, respectively. This is because EPI and CPI both learn *no AD function* like a_π and q_π (see the last column of Table 1). In CPI, $u_0, u_1, \cdots, u_m \in \mathcal{U}$ in the search space $\mathcal{X}\times\{u_j\}_{j=0}^m$ are any vectors in $\mathcal{U}$ such that $\operatorname{span}\{u_j - u_{j-1}\}_{j=1}^m = \mathbb{R}^m$. Denote $v_i \doteq q_i(\cdot,\pi_i(\cdot))/\kappa$ in the QPI case. Then, all of the four off-policy PI satisfy the following theorem.

Theorem 2. *The sequences $\{\pi_i\}_{i=0}^\infty$ and $\{v_i\}_{i=0}^\infty$ generated by any of the four off-policy PI under Assumptions 1, 2, and 3 satisfy* (P1)–(P4) *in Theorem 1. Moreover, for all $i \in \mathbb{N}\cup\{0\}$: $a_i = a_{\pi_i}$ (API), $q_i = q_{\pi_i}$ (QPI), and $c_i = c_{\pi_i}$ (CPI).*

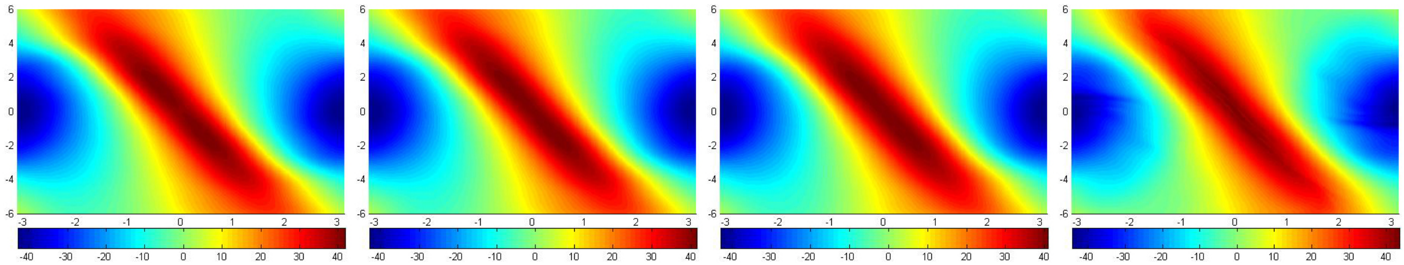
Table 1: Details about the (Partially) Model-Free Off-policy PI methods (API, QPI, EPI, and CPI)

Name	Policy Evaluation	Policy Improvement	Constraint(s)	Search Space	Fnc(s) to be Estimated
API	$\mathbb{E}_\mu^{(x,u)}\big[D_\gamma(v_i)\big] = \mathbb{E}_\mu^{(x,u)}\big[\mathcal{I}_\gamma\big(R - a_i + a_i^{\pi_i}\big)\big]$	$\pi_{i+1}(x) \in \arg\max_{u\in\mathcal{U}} a_i(x,u)$	$a_i^{\pi_i}(x) = 0$	$\mathcal{X}\times\mathcal{U}$	v_π and a_π
QPI	$\mathbb{E}_\mu^{(x,u)}\big[D_\beta\big(q_i^{\pi_i}\big)\big] = \kappa\cdot\mathbb{E}_\mu^{(x,u)}\big[\mathcal{I}_\beta\,(R - q_i)\big]$	$\pi_{i+1}(x) \in \arg\max_{u\in\mathcal{U}} q_i(x,u)$	(none)	$\mathcal{X}\times\mathcal{U}$	q_π
EPI	$\mathbb{E}_\mu^x\big[D_\gamma(v_i)\big] = \mathbb{E}_\mu^x\big[\mathcal{I}_\gamma\big(R^{\pi_i} - \nabla v_i\,(f_c - f_c^{\pi_i})\big)\big]$	exactly same as that in Alg. 1	(none)	$\mathcal{X}$	v_π
CPI	$\mathbb{E}_\mu^{(x,u)}\big[D_\gamma\big(v_i\big)\big] = \mathbb{E}_\mu^{(x,u)}\big[\mathcal{I}_\gamma\big(R^{\pi_i} - c_i\,\xi_\cdot^{\pi_i}\big)\big]$	$\pi_{i+1}(x) = \sigma(c_i(x))$	(C1) & (C2)	$\mathcal{X}\times\{u_j\}$	v_π and c_π

5 Simulation Examples: Applications to a Swing-up Pendulum Task

We simulated the proposed PI methods with the 2nd-order inverted-pendulum model ($n = 2$ and $m = 1$): $\dot\theta_\tau = w_\tau$ and $J\dot w_\tau = -\varrho\, w_\tau + mgl\sin\theta_\tau + U_\tau$, where θ_τ and w_τ are state variables representing the angular position and velocity of the pendulum at time $\tau \geq 0$, and U_τ is the external torque input to the pendulum at time τ limited as $|U_\tau| \leq U_{\mathsf{max}}$ for $U_{\mathsf{max}} = 5$ [N·m]. The physical parameters were set to $\varrho = 0.01$, $m = l = 1$, $g = 9.81$, and $J = ml^2 = 1$. The state and action spaces in this example are $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{U} = [-U_{\mathsf{max}}, U_{\mathsf{max}}]$; the state vector is $X_\tau \doteq [\,\theta_\tau \;\; w_\tau\,]^\mathsf{T} \in \mathcal{X}$; f in the dynamics is given by $f(x,u) = f_d(x) + F_c(x)u$ with $f_d(x) = [\,x_2 \quad (mgl\,\sin x_1 - \varrho x_2)/J\,]^\mathsf{T}$ and $F_c(x) = [\,0 \quad 1/J\,]^\mathsf{T}$ and thus satisfies (C1), where $x = [\,x_1 \;\; x_2\,]^\mathsf{T} \in \mathcal{X}$. Note that this inverted pendulum setting is exactly same to that in [2]; since the maximum torque U_{max} is smaller than mgl, the policy has to swing the pendulum several times to reach the upright position.

Our learning objective in the simulation is to make the pendulum swing up and eventually settle down at the upright position $\theta_{\mathsf{final}} = 2\pi k$ for some integer k. The reward R to achieve such a goal under the torque limit $|U_\tau| \leq U_{\mathsf{max}}$ was set to $R(x,u) = R_0(x) - S(u)$ with $R_0(x) = 10^2\cos x_1$ and $S(u) = \lim_{v\to u}\int_0^v \sigma^{-1}(\xi)\,d\xi$, where $\sigma(\xi) = U_{\mathsf{max}}\tanh(\xi/U_{\mathsf{max}})$ is a sigmoid function that saturates at $\pm U_{\mathsf{max}}$. Notice that our setting satisfies (C1) and (C2) both of which are necessary to simulate CPI with its simple policy improvement $\pi_{i+1}(x) = \sigma(c_i(x))$; both also simplify the policy improvement of PI and EPI



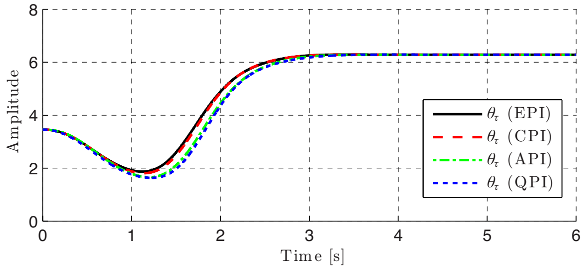
(a) $\hat{v}_{10}(x)$ (EPI) (b) $\hat{v}_{10}(x)$ (CPI) (c) $\hat{v}_{10}(x)$ (API) (d) $\hat{q}_{10}(x, \hat{\pi}_{10}(x))/\kappa$ (QPI)

Figure 1: The estimates of $v_i|_{i=10}$ ($\approx v_*$) done by the off-policy PI methods over the region Ω_{x}; the horizontal and vertical axes correspond to the values of the angular position x_1 and the velocity x_2 of the pendulum; $\hat{v}_i$, $\hat{q}_i$, and $\hat{\pi}_i$ denote the estimates of v_i, q_i, and π_i obtained by running each method. Note that $v_i = q_i(\cdot, \pi_i(\cdot))/\kappa$ in QPI.

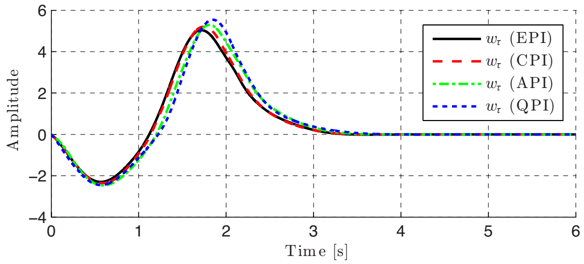
as $\pi_{i+1}(x) = \sigma(F_c^{\mathsf{T}}(x)\nabla v_i^{\mathsf{T}}(x))$, but not of API and QPI at all. By integration by parts and $\tanh^{-1}(u/U_{\mathsf{max}}) = \frac{1}{2}\ln(u_+/u_-)$, where $u_\pm \doteq 1 \pm u/U_{\mathsf{max}}$, the action penalty $S(u)$ is explicitly expressed as $S(u) = (U_{\mathsf{max}}^2/2) \cdot \ln\left(u_+^{u_+} \cdot u_-^{u_-}\right)$ which is finite over $\mathcal{U}$ and has its minimum ($= 0$) at $u = 0$ and its maximum ($\approx$ 17.3287) at $u = \pm U_{\mathsf{max}}$. This establishes the boundedness of the reward R over $\mathcal{X} \times \mathcal{U}$ and thereby, admissibility and boundedness of v_π for any policy π (see Section 2).

The initial policy π_0 and the parameters in all of the methods were set to $\pi_0 = 0$, $\gamma = 0.1$, $\Delta t = 10$ [ms], and in QPI, $\beta = 1$. The behavior policy μ used in the off-policy simulations was $\mu = 0$ (EPI) and $\mu(t, x, u) = u$ (API, QPI, and CPI). In API and QPI, the next target policy π_{i+1} is given by $\pi_{i+1}(x) \approx \sigma(y_i(x))$, where $y_i(x)$ is the output of a radial basis function network (RBFN) to be trained by policy improvement using a_i and q_i, respectively. The functions v_i, a_i, q_i, and c_i were all approximated by RBFNs as well. Instead of the whole spaces $\mathcal{X}$ and $\mathcal{X} \times \mathcal{U}$, we considered their compact regions $\Omega_{\mathsf{x}} \doteq [-\pi, \pi] \times [-6, 6]$ and $\Omega_{\mathsf{x}} \times \mathcal{U}$; since our inverted-pendulum system and the VF are 2π-periodic in the angular position x_1, the state value $x \in \mathcal{X}$ was normalized to $\bar{x} \in [-\pi, \pi] \times \mathbb{R}$ whenever input to the RBFNs. Further details about the RBFNs and the implementation methods are all omitted for brevity; the result of PI (Algorithm 1) is also omitted since it is almost exactly same to that of EPI.

Fig. 1 shows the estimated values of v_{π_i} over Ω_{x} at $i = 10$. Here, v_{π_i} can be considered to be approximately equal to the optimal one v_* after convergence. As shown in Fig. 1, the landscapes of the final VF estimates generated by different PI methods are all consistent and approximately equal to each other. The same goes for the state trajectories in Fig. 2 generated under the estimated policy $\hat{\pi}_i$ of π_i finally obtained at $i = 10$ by each off-policy method. They also all achieved the learning objective with $\theta_{\mathsf{final}} = 2\pi$ at around $t = 3$ [s]. Note that in our case, the initial policy $\pi_0 = 0$ was not asymptotically stabilizing while it should be in the PI methods *under the optimal control setting (without discounting)* to achieve such learning objective [3, 4].



(a) $\mathbb{E}_{\hat{\pi}_{10}}[\theta_\tau | X_0 = x_0]$, the trj. of the angular position θ_τ



(b) $\mathbb{E}_{\hat{\pi}_{10}}[w_\tau | X_0 = x_0]$, the trj. of the angular velocity w_τ

Figure 2: State trjs. under $x_0 = [\,1.1\pi \;\; 0\,]^{\mathsf{T}}$ and $\hat{\pi}_{10}$.

References

[1] L. C. Baird III. Advantage updating. Technical report, DTIC Document, 1993.

[2] K. Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

[3] J. Y. Lee, J. B. Park, and Y. H. Choi. Integral reinforcement learning for continuous-time input-affine nonlinear systems with simultaneous invariant explorations. *IEEE Trans. Neural Networks and Learning Systems*, 26(5):916–932, 2015.

[4] F. L. Lewis and D. Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE Circuits and Systems Magazine*, 9(3):32–50, 2009.

[5] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. Second Edition in Progress, MIT Press, Cambridge, MA, 2017.

Automatically Deriving Rational Heuristics for Risky Choice

Falk Lieder [1]
Dept. of Psychology, UC Berkeley
falk.lieder@berkeley.edu

Paul M. Krueger [1]
Dept. of Psychology, UC Berkeley
pmk@berkeley.edu

Thomas L. Griffiths
Dept. of Psychology, UC Berkeley
tom_griffiths@berkeley.edu

[1] **These authors contributed equally.**

Abstract

What is the optimal way to make a decision given that your time is limited and your cognitive resources are bounded? To address this question, we formalized the bounded optimal decision process as the solution to a meta-level Markov decision process whose actions are costly computations. We approximated the optimal solution and evaluated its predictions against human choice behavior in the Mouselab paradigm, which is widely used to study decision strategies. Our computational method rediscovered well-known heuristic strategies, such as Take-The-Best (TTB), and it also discovered a novel, previously unknown heuristic that integrates TTB with satisficing (SAT-TTB). An experiment using the Mouselab paradigm confirmed that people do indeed use SAT-TTB on a non-negligible fraction of problems—especially when the stakes are low. Furthermore, our model made three predictions about when people should use which kind of decision strategy: First, our model predicts that people should use fast-and-frugal heuristics more frequently when one outcome is much more likely than the others. Second, our model predicts that people should use simple heuristics, like TTB, SAT-TTB, and random choice, primarily when the stakes are low. Third, our model predicts that when the stakes are high people should invest more time and effort to reap a higher fraction of the highest possible expected payoff. Our participants' clicks and decisions in the Mouselab experiment confirmed all three of these predictions. These findings are a proof-of-concept that optimal cognitive strategies can be automatically derived as the rational use of finite time and bounded cognitive resources.

Keywords: risky choice; heuristics; bounded rationality; meta-decision making; meta-level reinforcement learning

Acknowledgements

This work was supported by grant number ONR MURI N00014-13-1-0341 and a grant from the Templeton World Charity Foundation.

1 Introduction and Background

People and robots have to make decisions in a limited amount of time and with bounded cognitive resources. Given that these resources are scarce, which strategy should a decision-maker employ to use its resources most effectively? The theory of bounded optimality and rational metareasoning (Russell & Wefald, 1991; Russell & Subramanian, 1995) was developed to answer this question for rational agents with limited performance hardware. It frames this problem as selecting computations so as to maximize the sum of the rewards of resulting decisions minus the costs of the computations involved.

Concretely, the problem of choosing computations optimally can be formulated as a meta-level Markov decision process (meta-level MDP; Hay, Russell, Tolpin, & Shimony, 2012). A meta-level MDP

$$M_{\text{meta}} = (\mathcal{B}, \mathcal{C}, T_{\text{meta}}, r_{\text{meta}}) \tag{1}$$

is a Markov decision process whose actions $\mathcal{C}$ are cognitive operations, its states $\mathcal{B}$ represent the agent's beliefs, and the transition function T_{meta} models how cognitive operations change the agent's beliefs. In addition to a set of computations C that update the agent's belief, the cognitive operations also include the meta-level action $\perp$ that terminates deliberation decides based on the current belief b_t. The belief state b_t encodes the agent's probabilistic beliefs about the domain it is reasoning about. The meta-level reward function r_{meta} captures the cost of thinking (Shugan, 1980) and the reward r the agent expects to receive from the environment when it stops deliberating and takes action. The computations C do not yield any external reward. Their only effect is to update the agent's beliefs. Hence, the meta-level reward for performing a computation $c \in C$ is $r_{\text{meta}}(b_t, c) = -\text{cost}(c)$. By contrast, terminating deliberation and taking action ($\perp$) does not update the agent's belief. Instead, its value lies in the anticipated reward for taking action, that is $r_{\text{meta}}(b_t, \perp) = \arg\max_a b_t^{(\mu)}(a)$, where $b_t^{(\mu)}(a)$ is the expected reward of taking action a according to the belief b_t.

Consistent with rational metareasoning, people flexibly adapt their decision processes to the structure of the problem they face. Concretely, Payne, Bettman, and Johnson (1988) found that people use fast-and-frugal heuristics, like Take-The-Best (TTB, which chooses the option favored by the most predictive attribute and ignores all other attributes), more frequently under time pressure and when one outcome is much more likely than the others. In this research, participants choose between the gambles $g_1, \cdots, g_n$ which are defined by the payoffs they assign to each of four possible outcomes with known probabilities ($P(O)$). Participants could inspect a payoff matrix $V_{o,g}$ with one row for each outcome o and one column for each gamble g. Critically, each payoff is only revealed when the participant clicks on the corresponding cell of the payoff matrix using a mouse; this task is hence referred to as the *Mouselab* paradigm (see Figure 1).

The adaptiveness of people's strategy choices in the Mouselab paradigm suggests that their decision processes are efficient and effective. But it is difficult to test whether they are optimal, because it is unclear what it means to decide optimally when one's time is valuable and one's cognitive resources are limited. To clarify this, the following section develops a normative theory of resource-bounded decision making in the Mouselab paradigm.

2 Boundedly-optimal decision-making

To model the meta-decision problem posed by the Mouselab task, we characterize the decision-maker's belief state b_t by probability distributions on the expected values $e_1 = \mathbb{E}[v_{O,g_1}], \cdots e_n = \mathbb{E}[v_{O,g_n}]$ of the n available gambles $g_1, \cdots, g_n$. Furthermore, we assume that for each element $v_{o,g}$ of the payoff matrix V there is one computation $c_{o,g}$ that inspects the payoff $v_{o,g}$ and updates the agent's belief about the expected value of the inspected gamble according to Bayesian inference. Since the entries of the payoff matrix are drawn from the normal distribution $\mathcal{N}(\bar{v}, \sigma_v^2)$, the resulting posterior distributions are also Gaussian. Hence, the decision-maker's belief state b_t can be represented by $b_t = (b_{t,1}, \cdots, b_{t,n})$ with

$$b_{t,g} = \left(b_{t,g}^{(\mu)}, b_{t,g}^{(\sigma^2)}\right), \tag{2}$$

where $b_{t,g}^{(\mu)}$ and $b_{t,g}^{(\sigma^2)}$ are the mean and the variance of the probability distribution on the expected value of gamble g of the belief state b_t. Given the set $\mathcal{O}_t$ of the indices $(k_o^{(1)}, k_g^{(1)}), \cdots, (k_o^{(t)}, k_g^{(t)})$ of the t observations made so far, the means and variances characterizing the decision-maker's beliefs are given by

$$b_{t,g}^{(\mu)} = \sum_{(o,g)\in\mathcal{O}} p(o) \cdot v_{o,g} + \sum_{(o,g)\notin\mathcal{O}} p(o) \cdot \bar{v} \tag{3}$$

$$b_{t,g}^{(\sigma^2)} = \sum_{(o,g)\notin\mathcal{O}} p(o)^2 \cdot \sigma_v^2. \tag{4}$$

The meta-level transition function $T(b_t, c_{o,g}, b_{t+1})$ encodes the probability distribution on what the updated means and variances will be given the observation of a payoff value $V_{o,g}$ sampled from $\mathcal{N}(\bar{v}, \sigma_v^2)$. The meta-level reward for performing the computation $c_{o,g} \in C$ encodes that acquiring and processing an additional piece of information is costly. We assume that the cost of all such computations is an unknown constant λ.The meta-level reward for terminating deliberation and taking action is $r_{\text{meta}}(b_t, \perp) = \max_g b_t^{(\mu)}(g)$.

2.1 Approximating the optimal meta-level policy: Bayesian value function approximation

Unfortunately, computing the optimal policy for the meta-level MDP defined above is intractable. However, it can be approximated using methods from reinforcement learning. We initially used the semi-gradient SARSA algorithm (Sutton & Barto, 1998) with limited success. We therefore developed a new algorithm that replaces the gradient descent component of that algorithm by Bayesian linear regression.

Our algorithm learns a linear approximation to the meta-level Q-function

$$Q_{\text{meta}}(b, c) \approx \sum_k w_k \cdot f_k(b, c), \tag{5}$$

whose features $\mathbf{f}$ include a constant, features of the belief state b_t, and features of the computation c_t. The features of the belief state were the expected value of the maximum of the gambles' expected values ($\mathbb{E}\left[\max_g E_g | b_t\right]$), the decision-maker's uncertainty about it ($\sqrt{\text{Var}[\max_g E_g | b_t]}$), the largest posterior mean ($\max_g b_{t,g}^{(\mu)}$) and its associated uncertainty ($\sqrt{\mu_{t,g^\star}^{(\sigma^2)}}$ where $g^\star = \arg\max_g b_{t,g}^{(\mu)}$), the second largest posterior mean and the decision-maker's uncertainty about it, and the expected regret $\mathbb{E}\left[\text{regret}(g) | b_t\right]$ that the decision-maker would experience if they chose based on their current belief (where $\text{regret}(g) = \max_g E_g - \max_g b_{t,g}^{(\mu)}$ for $E_i \sim \mathcal{N}(b_{t,i}^{(\mu)}, b_{t,i}^{(\sigma)})$ for all gambles i). The features of the computation $c_{o,g}$ were its myopic value of computation ($\text{VOC}(b_t, c_{o,g})$; see Russell & Wefald, 1991), the current uncertainty about the expected value of the inspected gamble ($b_{t,g}^{(\sigma)}$), the probability of the inspected outcome $P(o)$, the difference between the largest posterior mean and the posterior mean of the inspected outcome, a binary variable indicating whether the computation acquired a new piece of information, and the expected reduction in regret minus its cost $\mathbb{E}\left[\mathbb{E}[\text{regret}(\arg\max_g b_{t,g}^{(\mu)}) | B_{t+1}] - \mathbb{E}[\text{regret}(\arg\max_g b_{t,g}^{(\mu)}) | b_t]\right] - \lambda$, where the random variable B_{t+1} denotes the belief state resulting from performing computation $c_{o,g}$.

The weights $\mathbf{w}$ are learned by Bayesian linear regression of the bootstrap estimate $\hat{Q}(b, c)$ of the meta-level value function onto the features $\mathbf{f}$. The bootstrap estimator is

$$\hat{Q}(b_t, c_t) = r_{\text{meta}}(b_t, c_t) + \hat{w}_t' \cdot \mathbf{f}(b_{t+1}, c_{t+1}), \tag{6}$$

where $\hat{w}_t$ is the posterior mean on the weights w given the observations from the first t trials, and $\mathbf{f}(b_{t+1}, c_{t+1})$ is the feature vector characterizing the subsequent belief state b_{t+1} and the computation c_{t+1} that will be selected in it.

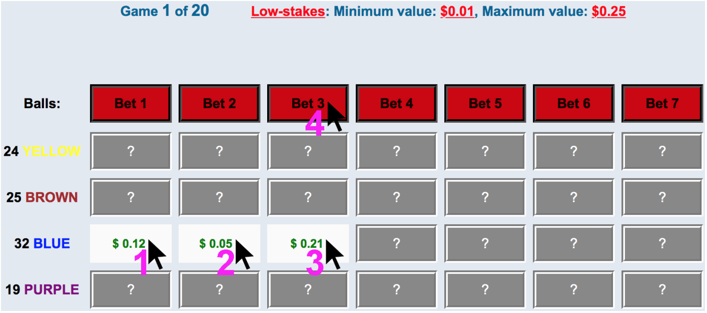
Given the learned posterior distribution on the feature weights $\mathbf{w}$, the next computation c is selected by contextual Thompson sampling (Agrawal & Goyal, 2013). Specifically, to make the tth meta-decision, a weight vector $\tilde{w}$ is sampled from the posterior distribution of the weights given the series of meta-level states, selected computations, and the resulting value estimates experienced so far, that is $\tilde{w} \sim P(\mathbf{w} | (b_1, c_1, \hat{Q}(b_1, c_1)), \cdots, (b_{k-1}, c_{k-1}, \hat{Q}(b_{k-1}, c_{k-1})))$. The sampled weight vector $\tilde{w}$ is then used to predict the Q-values of each available computation $c \in \mathcal{C}$ according to Equation 5. Finally, the computation with the highest predicted Q-value is selected.

2.2 Application to Mouselab experiment

As a proof of concept, we applied our approach to the Mouselab experiment described below. The experiment comprises 50% high-stakes problems (reward range: $\$0.01-\$9.99$) and 50% low-stakes problems (reward range: $\$0.01-\$0.25$). Since participants are informed about the stakes, we learned two separate policies for high-stakes and low-stakes problems, respectively. Half of each of those problems had nearly uniform outcome probabilities ("low dispersion") and for the other half one outcome was much more likely than all others combined ("high dispersion"). The parameters of the simulated environment were exactly equal to those of the experiment described below. Our model assumed that people play each game as if they receive the payoff of the selected gamble. We estimated the cost per click to be about $\lambda = 3$ cents. This value was selected to roughly match the average number of acquisitions observed in the experiment. To approximate the optimal meta-decision policy for this task, we ran our feature-based value function approximation method for 4000 low-stakes training trials and 4000 high-stakes training trials, respectively.

2.3 Model predictions

The optimal solution approximated by our method optimally trades off decision quality against the (time) cost of reaching the decision. This is why the learned policy prioritizes probable outcomes over less probable outcomes and invests more computation when the stakes are high. Our method automatically discovered strategies that people are known to use in the Mouselab paradigm as well as a novel strategy that has not been reported yet. Our method rediscovered TTB, Weighted-Additive Strategy (WADD, which computes all gambles expected values based on all possible payoffs), and the random choice strategy. In addition, it discovered a new hybrid strategy that combines TTB with satisficing (SAT-TTB). Like TTB, SAT-TTB inspects only the payoffs for the most probable outcome. But unlike TTB and like SAT, SAT-TTB terminates as soon as it finds a gamble whose payoff for the most probable outcome is high enough. On average, this value was about $\$0.15$ when the payoffs ranged from $\$0.01$ to $\$0.25$ (i.e., low-stakes trials). Figure 1 illustrates this strategy.



Game 1 of 20 Low-stakes: Minimum value: $0.01, Maximum value: $0.25

Balls:	Bet 1	Bet 2	Bet 3	Bet 4	Bet 5	Bet 6	Bet 7
24 YELLOW	?	?	?	?	?	?	?
25 BROWN	?	?	?	?	?	?	?
32 BLUE	$ 0.12	$ 0.05	$ 0.21	?	?	?	?
19 PURPLE	?	?	?	?	?	?	?

Figure 1: The Mouselab paradigm, showing an example sequence of clicks generated by the SAT-TTB strategy, which was discovered through meta-level RL.

Furthermore, our model makes intuitive predictions about the contingency of people's choice processes on stakes and outcome probabilities. First, our model predicts that people should use fast-and-frugal heuristics more frequently in high-dispersion trials. Concretely, our model generated TTB as the strategy of choice for 100% of the high-dispersion problems with low-stakes, but for low-dispersion problems with low-stakes the model considered the random choice strategy to be optimal in the majority (56%) of cases; it used the SAT-TTB hybrid strategy for 24% of such trials, and it indicated the TTB strategy for the remaining 20%. Second, our model predicts that people should use simple heuristics, like TTB, SAT-TTB, and random choice, primarily when the stakes are low. Our model used these heuristics for 100% of the low-stakes problems. But for high-stakes problems, the model never used any of these or other frugal strategies. Instead, the model typically inspected the vast majority of all cells ($24.8/28$ for low-dispersion problems and $23.7/28$ for high-dispersion problems). Third, our model predicts that when the stakes are high people should invest more time and effort ($F(1, 396) = 9886.8, p < 0.0001$) to reap a higher fraction of the highest possible expected payoff ($F(1, 339) = 135.24, p < 0.0001$).

3 Experimental test of novel predictions

To test the predictions of our model, we conducted a new Mouselab experiment that manipulated the stakes and dispersion of outcome probabilities within subjects in an identical manner to the model simulations.

3.1 Methods

We recruited 200 participants on Amazon Mechanical Turk. The experiment took about 30 minutes and participants received base pay of $\$1.50$ and one of their twenty winnings was selected at random and awarded as a bonus (mean bonus = $\$3.53$). Participants performed a variation of the Mouselab task (Payne et al., 1988). They played a series of 20 games divided into two blocks. Figure 1 shows a screenshot of one game. Every game began with a 4×7 grid of occluded payoffs: there were seven gambles to choose from (columns) and four possible outcomes (rows). The occluded value in each cell specified how much the gamble indicated by its column would pay if the outcome indicated by its row occurred. The outcome probabilities were described by the number of balls of a given color in a bin of 100 balls, from which the outcome would be drawn. For each trial, participants were free to inspect any number of cells before selecting a gamble, with no time limit. The value of each inspected cell remained visible onscreen for the duration of the trial. Upon selecting a gamble, the resulting reward was displayed.

The experiment used a 2×2 within subjects design. Each block of ten trials was either low-stakes or high-stakes, with block order randomly counterbalanced across participants. In games with low-stakes, the possible outcomes ranged from $\$0.01$ to $\$0.25$, while in high-stakes games, outcomes ranged from $\$0.01$ to $\$9.99$. Within each block, there were five low-dispersion trials and five high-dispersion trials, ordered randomly. In low-dispersion trials, the probability of each of the four outcomes ranged from 0.1 to 0.4, whereas in high-dispersion trials, the probability of the most likely outcome ranged from 0.85 to 0.97.

We identified six different decision strategies, in humans and in simulations, using the following definitions: TTB was defined as inspecting all cells in the row corresponding to the most probable outcome and nothing else. SAT occurs when one gamble's payoffs are inspected for all four outcomes, potentially followed by the inspection of all outcomes of another gamble, and so on, but leaving at least one gamble unexamined. The hybrid strategy, SAT-TTB, was defined as inspecting the payoffs of 1 to 6 gambles for the most probable outcome and not inspecting payoffs for any other outcome. WADD was defined as inspecting all 28 cells column by column. Random decisions mean zero samples were taken.

3.2 Results

Our process tracing data confirmed that people do indeed use the SAT-TTB strategy discovered by our model. Table 1 shows the frequency of various decision strategies, for each of the four different types of trials. Out of 4000 trials across all participants, TTB was the most common strategy overall, accounting for 25.3% of all trials. SAT-TTB was the second most common strategy among those we examined: participants employed this strategy on 10.7% of all trials. In 8.0% of trials participants chose randomly without making any observations—mostly during low-stakes games. Interestingly, we also observed a second novel strategy that we call Take-The-Best-Two (TTB2). This strategy inspects all gambles' payoffs for the *two* most probable outcomes (and nothing else), and was used in 6.3% of trials. The WADD strategy occurred in 4.5% of trials. Finally, the SAT strategy was used in 3.1% of games.

Consistent with our model's first prediction, people used TTB more frequently when the dispersion was high ($\chi^2(1) = 897.9, p < 0.0001$). Consistent with our model's second prediction, participants used simple heuristics more frequently when the stakes were low: the frequency of the random choice—the simplest heuristic—increased significantly from 4.2% on high-stakes problems to 19.9% on low-stakes problems ($\chi^2(1) = 88.2, p < 0.0001$), and so did the frequency of the second simplest heuristic, SAT-TTB ($\chi^2(1) = 86.3, p < 0.0001$), and the third simplest heuristic, TTB ($\chi^2(1) = 20.0, p < 0.0001$). An increase in frequency of SAT from high- to low-stakes games was nearly significant ($\chi^2(1) = 3.4, p = 0.07$). Finally, consistent with our model's third prediction, the frequency of the most effortful and most accurate strategy, WADD, increased with the stakes ($\chi^2(1) = 19.3, p < 0.0001$).

4 Conclusion

In summary, our resource-rational theory of multi-alternative risky choice predicted some of the main strategies people use in the Mouselab paradigm and the conditions under which they are selected. In addition to automatically discovering known strategies and contingencies, our computational approach also discovered a novel, previously unknown heuristic that integrates TTB with satisficing (SAT-TTB), and our experiment confirmed that people do indeed use SAT-TTB on a non-negligible fraction of problems—especially when the stakes are low.

Tajima, Drugowitsch, and Pouget (2016) solved meta-level MDPs to derive boundedly optimal drift-diffusion models. The strategy discovery method presented here generalizes this approach to more complex decision mechanisms that can process and generate evidence in many different ways.

	Frequency				
Strategy	Total	HS-HD	HS-LD	LS-HD	LS-LD
TTB	1012	392	64	449	107
SAT-TTB	412	68	54	140	150
Random	320	41	42	111	126
TTB2	251	34	94	25	98
WADD	178	33	84	19	42
SAT	89	14	22	23	30

HS-HD = High-stakes, high-dispersion HS-LD = High-stakes, low-dispersion
LS-HD = Low-stakes, high-dispersion LS-LD = Low-stakes, low-dispersion

Table 1: Frequency of strategy types for each type of trial.

One limitation of the current work is that we do not know how closely our algorithm approximated the optimal policy, and it is possible that a more accurate approximation would yield somewhat different predictions. Future work will systematically evaluate the accuracy of our approximation method on smaller problems for which the optimal meta-level policy can be computed exactly.

Our proof-of-concept study suggests that formulating the problem of making optimal use of finite time and limited cognitive resources as a meta-level MDP is a promising approach to discovering cognitive strategies. This approach can be leveraged to develop more realistic normative standards of human rationality. This might enable future work to systematically evaluate the extent to which people are resource-rational. In the long term, our approach could be used to improve human reasoning and decision-making by discovering rational heuristics and teaching them to people.

References

Agrawal, S., & Goyal, N. (2013). Thompson sampling for contextual bandits with linear payoffs. In *Proceedings of the 30th international conference on machine learning* (pp. 127–135).

Hay, N., Russell, S., Tolpin, D., & Shimony, S. (2012). Selecting computations: Theory and applications. In N. de Freitas & K. Murphy (Eds.), *Uncertainty in artificial intelligence: Proceedings of the twenty-eighth conference.* Corvallis, OR: AUAI Press.

Payne, J. W., Bettman, J. R., & Johnson, E. J. (1988). Adaptive strategy selection in decision making. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *14*(3), 534.

Russell, S. J., & Subramanian, D. (1995). Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, *2*, 575–609.

Russell, S. J., & Wefald, E. (1991). Principles of metareasoning. *Artificial Intelligence*, *49*(1-3), 361–395.

Shugan, S. M. (1980). The cost of thinking. *Journal of consumer research*, *7*(2), 99–111.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1) (No. 1). MIT press Cambridge.

Tajima, S., Drugowitsch, J., & Pouget, A. (2016). Optimal policy for value-based decision-making. *Nature communications*, *7*.

SAIL: A Temporal Difference Approach to State Aware Imitation Learning

Yannick Schroecker
College of Computing
Georgia Institute of Technology
yannickschroecker@gatech.edu

Charles L. Isbell
College of Computing
Georgia Institute of Technology
isbell@cc.gatech.edu

Abstract

Imitation learning aims at training agents to reproduce a teachers policy based on a set of demonstrated states and actions. However, attempting to reproduce actions without learning about the environment can lead the agent to situations that are unlike the ones encountered as part of the provided demonstrations, making it more likely for the agent to make a mistake. In this work we present State Aware Imitation Learning (SAIL), an algorithm for imitation learning which augments the supervised approach to imitation learning by explicitly trying to remain in states that are similar to the states seen as part of a demonstration and in which the agent knows what action to choose. The algorithm achieves this goal by maximizing the joint likelihood over states and actions at each time step. Based on existing work by Morimura et al. [6], we show that an update rule similar to online temporal difference learning can be used to learn the gradient of said joint distribution which allows us to perform gradient ascent. The resulting policy allows the agent to remain close to states in which it knows what to do which prevents errors from accumulating over time. Naturally, learning this gradient requires additional information about the world which can be obtained through unsupervised interactions with the environment. However, it does not require further input from the teacher. While the algorithm proposed in this paper can be used with any kind of state space, we evaluate our approach on a simple race track domain with 7425 discrete states. Using a tabular representation combined with randomness makes it impossible to train a policy in a purely supervised way such that it behaves near optimally in states that have not been encountered as part of a demonstration. We show that using unsupervised sample transitions with our approach allows the agent to learn a reasonable policy outside of the set of observed states and show that SAIL outperforms a purely supervised learning approach on this task.

Keywords: Imitation Learning, Learning from Demonstration, Temporal Difference Learning, Policy Evaluation

Acknowledgements

This work was supported by the Office of Naval Research under grant N000141410003

1 Introduction

Learning to act in an unknown environment can be considered a primary focus in the field of Artificial Intelligence and has been approached from many different directions. Solutions in the field of imitation learning are attempting to solve this problem by learning from a teacher and have, among others, been successfully applied in domains such as robotics[2] or decision making in games (e.g. [10]). A common and straight-forward approach is to train a control policy using a supervised Learning from Demonstration (LfD) approach. In this approach a mapping from states to actions is learned directly with supervised learning. However, this relies on the learning algorithm to be able to generalize to all the states the agent might possibly see and errors can accumulate when the agents actions lead it away from the demonstrated trajectories[7, 8]. In this work we aim to alleviate this problem and present State Aware Imitation Learning (SAIL), an algorithm which, based on sample roll-outs, learns how to stick to states that are similar to the demonstrated ones while simultaneously learning to reproduce the teachers actions in those states. In short, the goal is for the agent to find states in which it knows what to do and then do it. To achieve this, we find the parameters θ of a parametric policy π_θ by maximizing the likelihood $\max_\theta p(s_1, a_1, s_2, a_2, \dots | \theta)$ of the demonstrated trajectories (consisting of states $s_1, s_2, \dots$ and actions $a_1, a_2, \dots$). We make the assumption that the policy leads to a stationary distribution over states and simplify the learning problem by assuming independence between time steps. The learning problem is then given by

$$\hat{\theta} = \text{argmax}_\theta p(s_1, a_1, s_2, a_2, \dots | \theta) = \text{argmax}_\theta \sum_{i=0}^{N} \log p(a_i | s_i, \theta) + \log p(s_i | \theta) \tag{1}$$

for N demonstrated transitions. Note that the first part of this sum corresponds to regular maximum likelihood based supervised learning and that $p(a_i|s_i)$ is given by the policy $\pi_\theta(a_i|s_i)$. The additional term $p(s_i|\theta)$ guides the agent towards the desired states and, under the above mentioned simplifying assumptions, is modeled by the stationary distribution $d^{\pi_\theta}(s)$. To maximize this log joint likelihood by gradient descent we now need to estimate $\nabla_\theta \log d^{\pi_\theta}(s)$. Note that this gradient is based on reproducing states and thus requires the agent to learn about the environment. Thus, if no transition model is given, additional sample roll-outs will be necessary to learn this gradient. We base our approach on the work of Morimura et al.[6] who introduce a least squares temporal difference approach to estimating this gradient and utilize it to derive a policy gradient approach in the reinforcement learning setting. We first extend this approach by arguing convergence for an online update rule and then use the learned gradient for imitation learning.

2 Related Works

The key problem with purely supervised approaches to imitation learning is that the agent may visit states unlike the ones seen during demonstration. Prior approaches to solving this problem are SMILe[8] and, building on it, DAGGER[9]. Ross et al. address the issue at hand by iteratively collecting roll-outs from the trained policy and collecting new demonstrations on the observed states. They show that their approach minimizes the classification error under the distribution of states seen by the agent when executing the learned policy as opposed to the distribution of states demonstrated by the teacher which ensures that the agent knows how to act in the states it will see during execution. However, while this algorithm works well and comes with theoretical guarantees, acquiring new demonstrations online during execution is often not possible. This is especially true if the demonstrations have been acquired by observation or if demonstrating the optimal action forces the agent to take that action during training. In this paper we aim to address the problem of visiting states without demonstrations when given a fixed set of demonstrations up front.

The second main approach related to SAIL is Inverse Reinforcement Learning (IRL). IRL methods use demonstrations in order to learn a reward function. Learning a reward function can either be an end on its own or be used to train a policy that may exhibit good behavior in states that supervised approaches cannot generalize to. IRL approaches have several similarities with SAIL. First of all, IRL methods use sample roll-outs or knowledge of the domains transition model in order to act in the presumed interest of the teacher, even in states for which the teacher has not provided demonstrations. Furthermore, many IRL approaches such as the popular Maximum Entropy Inverse Reinforcement Learning[12] explicitly reason over the distribution of states in order to learn a reward function. Note that the majority of IRL approaches require that a transition model is given but approaches based on sample roll-outs exist (e.g. [1, 5, 4]). However, despite the mentioned similarities, the policy learned by apprenticeship learning with IRL maximizes a different objective compared to the objective function considered in this paper. Also note that reward functions are not isomorphic to policies; multiple reward functions can yield the same policy and disambiguating between different possible reward functions is often a major focus of IRL approaches. Furthermore, and perhaps most importantly, IRL methods are limited by the representation of the reward function whose choice is usually further limited by the chosen IRL algorithm. SAIL does not need to explicitly represent the reward function and while the gradient still needs to be learned and represented, it is more similar to the representation of the policy and can be modeled by any differentiable parametric function.

Algorithm 1 State Aware Imitation Learning

```
function SAIL($\theta$, $\omega$, $\alpha_\theta$, $\alpha_\omega$, $\alpha_\mu$, $s_0$, $a_0$, ..., $s_T$, $a_T$)
    $\mu \leftarrow 0$
    for $k \leftarrow$ 0..#Iterations do
        for $i \leftarrow$ 0..#UnsupervisedSteps do
            $s, a, s' \leftarrow$ CollectTransition()
            $\omega \leftarrow \omega + \alpha_\omega \left(f_\omega(s) + \nabla_\theta \log \pi_\theta(a|s) - f_\omega(s')\right) \nabla_\omega f(s')$
            $\mu \leftarrow (1-\alpha_\mu)\mu + \alpha_\mu f(s')$
        for $t \leftarrow 0 \dots T$ do
            $\theta \leftarrow \theta + \alpha_\theta \left(\nabla_\theta \log \pi_\theta(a_t|s_t) + (f(s_t) - \mu)\right)$
    return $\theta$
```

3 Approach

In this section we describe State Aware Imitation Learning in detail. We first summarize the findings of Morimura et al.[6] in section 3.1, argue that convergence can be achieved for a stochastic gradient descent based temporal difference algorithm (in the linear case) in 3.2 and describe the full algorithm for imitation learning in 3.3.

3.1 Estimating $\nabla_\theta \log d^{\pi_\theta}(s)$ via temporal differences learning

Estimating the gradient of the logarithmic stationary state distribution based on a temporal-difference approach has first been proposed by Morimura et al.[6]. Based on the definition of the stationary state distribution, they derive a temporal difference equation. For all states s', we have:

$$\begin{aligned} \nabla_\theta d^{\pi_\theta}(s') &= \nabla_\theta \int d^{\pi_\theta}(s)\pi_\theta(a|s)p(s'|s,a)ds, a \\ \Rightarrow d^{\pi_\theta}(s')\nabla_\theta \log d^{\pi_\theta}(s') &= \int p(s,a,s')\left(\nabla_\theta \log d^{\pi_\theta}(s) + \nabla_\theta \log \pi_\theta(a|s)\right) ds, a \\ \Rightarrow \qquad 0 &= \int p(s,a|s')\left(\nabla_\theta \log d^{\pi_\theta}(s) + \nabla_\theta \log \pi_\theta(a|s) - \nabla_\theta \log d^{\pi_\theta}(s')\right) ds, a \end{aligned}$$

Note that this is an expected temporal difference error over the backwards transition probabilities $p(s'|s,a)$ and can be written as

$$\delta(s,a,s') := \nabla_\theta \log d^{\pi_\theta}(s) + \nabla_\theta \log \pi_\theta(a|s) - \nabla_\theta \log d^{\pi_\theta}(s')$$

Morimura et al. show that the expected sum of the temporal difference errors using this backwards Markov chain is equal to the expected sum of the temporal difference errors on the original, forward Markov chain defined by the transition probabilities $p(a,s'|s) = \pi(a|s)p(s'|s,a)$:

$$\sum_{t=0}^{\infty} E[\delta(s_t,a_t,s_{t+1})|p(a_t,s_{t+1}|s_t)] = \sum_{t=0}^{\infty} E[\delta(s_t,a_t,s_{t+1})|p(a_t,s_t|s_{t+1})]$$

This fact is then used to derive a least squares estimator of $\nabla_\theta \log d^{\pi_\theta}(s)$ using samples collected from the forward Markov chain. Note that in order to avoid ambiguity, an additional constraint has to be enforced which ensures that

$$E_s[\nabla_\theta \log d^{\pi_\theta}(s)] = 0 \tag{2}$$

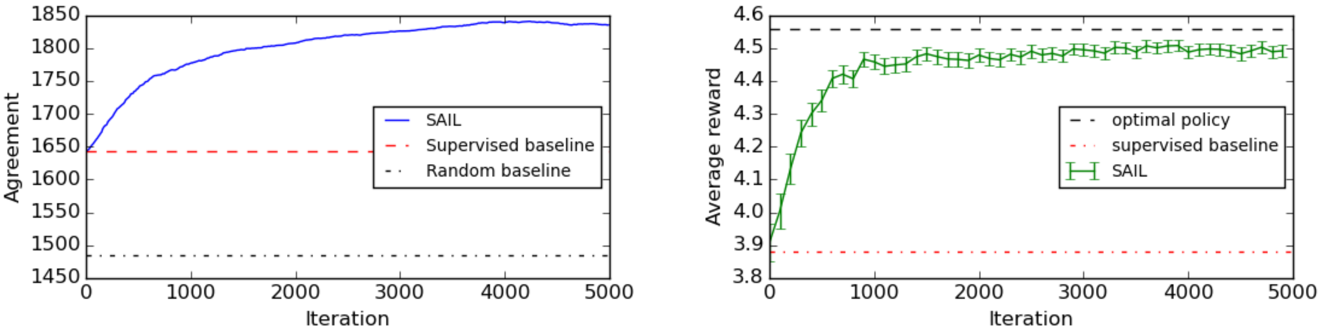
which is derived from the constraint $\int d^{\pi_\theta}(s)ds = 1$.

3.2 Estimating $\nabla_\theta \log d^{\pi_\theta}(s)$ with online temporal difference updates

Using the temporal difference error described above, we use modified online temporal difference learning to approximate $\nabla_\theta \log d^{\pi_\theta}(s)$. This increases computational efficiency of the algorithm and allows it to be applied to larger and higher dimensional state spaces. Let $f_\omega(s)$ be the current approximation of $\nabla_\theta \log d^{\pi_\theta}(s)$, the update equation will be analogous to the common temporal difference update and is given by

$$\omega_{k+1} = \omega_k + \alpha\nabla_\omega f(s')\left(f(s) + \nabla_\theta \log \pi(a|s) - f(s')\right)$$

To enforce the constraint $E_s[f(s)] = 0$, we run the algorithm until convergence and then project f onto the set of functions satisfying the specified constraint.



(a) (b)

Figure 1: a) Sum of probabilities of taking the optimal action. b) Reward $(+/-2\sigma)$ obtained during 5000 iterations of training

Convergence in the Linear Case We now consider the case of linear function approximation, we define $f_\omega(s) = \phi(s)^T\omega$ for a feature function $\phi(s)$. The update equation is then given by:

$$\omega_{k+1} = \omega_k + \alpha\phi(s')\left(\phi(s)^T\omega_k + \nabla_\theta \log\pi(a|s) - \phi(s')^T\omega_k\right)$$

The first thing we will point out is the relation to the established temporal difference learning algorithm in the average reward case. In this case a value function $V(s)$ is estimated based on observed rewards $r(s,a)$ using samples and an estimate of the mean reward $E[r(s,a)] \approx \mu$. The update is then given by $\omega_{k+1} = \omega_k + \alpha\nabla_\omega V_\omega(s)\left(V_{\omega_k}(s) + r(s,a) - \mu_k - V_{\omega_k}(s')\right)$. In our scenario, the reward is replaced by $\nabla_\theta \log\pi(a|s)$ where s and a are sampled from the forward Markov chain. Note that $E[\nabla_\theta \log\pi(a|s)] = 0$ and thus the average "reward" is 0. Convergence of average reward temporal difference learning has been thoroughly studied [11] and the only thing left to show is convergence properties are unchanged when sampling from the backwards Markov chain. This is easy to see as the proof by Tsitsiklis et al.[11] reduces the update to their steady state expectation, i.e. the expectation over $d^\pi(s)$. Morimura et al.[6] show that the stationary state distribution, assuming it exists, is the same for the forward and the backwards Markov chain. Thus the algorithm converges to an approximation $f(s) \approx \nabla_\theta \log d^\pi(s) + c$ where c is a constant vector of the same dimension as θ.

3.3 SAIL

Now we can use this learned and projected approximation of $\nabla_\theta \log d^{\pi_\theta}$ to maximize the joint distribution over demonstrated states and actions. Given a set of demonstrations, the full algorithm iteratively collects roll-outs, performs the temporal difference update described in section 3.2 and then follows the gradient to maximize the joint likelihood over all seen states and actions:

$$\nabla_\theta \log p(s_0, a_0, s_1, a_1, \ldots|\theta) = \sum_{t=0}^{T} \nabla_\theta \log\pi_\theta(a_t|s_t) + \left(f_\omega(s_t) - \sum_{t'=0}^{T} f_\omega(s_{t'})\right) \tag{3}$$

See algorithm 1 for a complete overview of SAIL.

4 Evaluation

In this section we evaluate the performance of an agent trained with SAIL compared to a supervised Learning from Demonstration approach. We use a variation of the racetrack domain introduced by Boularias et al.[1] in which the agent has to go around a track while managing momentum and the associated randomness. The randomness of this domain ensures that the agent will randomly visit states that have not been part of any demonstrations while its tabular nature ensures that it cannot use similarity between states to learn how to complete the task anyway. In order to reliably complete laps around the track, the agent has to learn how to recover when getting off the track. The domain consists of 33 by 9 possible positions, each of which have 25 states associated with it in order to describe the velocity (+2, +1, 0, -1 or -2) along both directions of the x-y-plane. This leads to a total of 7425 discrete states. The agent navigates in this domain by accelerating or decelerating, leading to 5 different actions: changing the velocity along the horizontal or the vertical axis by +1 or -1 and doing nothing. Randomness is introduced to the domain in the form of a possibility for failure in which case the velocities remain the same. Probability for failure is high (80%) if one of the velocities is +2 or -2 and low (10%) otherwise. The goal of the agent is to complete a lap while staying on track where being off track is defined as $x = 0$, $y = 0$, $x > 31$ or $y > 6$ as well being inside an inner rectangle defined by $2 < x < 31$ and $2 < y < 6$.

The agent learns to complete laps based on demonstrations given to it by an oracle. This oracle is defined by the optimal policy according to a reward of 5 for completing a lap, -5 for crossing the finish line the wrong way and -0.1 for being off track. We also use this reward to evaluate the performance of SAIL and the supervised LfD baseline. Note, however, that the reward is not being used in training SAIL or the LfD baseline and is not generally available when doing imitation learning. Supervised Learning from Demonstration in a tabular domain without noise works by first taking the same action as demonstrated in states where a demonstration exists and second taking actions according to a prior in states that haven't been part of a demonstration, e.g. in this case choosing actions uniformly. We compare this against SAIL after 5000 iterations where 20 trajectories are collected in an unsupervised way for each iteration. The temporal difference update is done with plain stochastic gradient descent and a learning rate of 0.1 whereas the gradient descent update on the policy is using a learning rate of 0.01. Figure 1b compares the average reward obtained by SAIL, supervised LfD and the optimal baseline and shows that, over time, the policy learned by SAIL approaches the reward obtained by the optimal policy and outperforms pure supervised learning alone meaning that the agent spends far less time off-track. As an additional evaluation, we compare on how many states the policies learned by SAIL and the baseline agree with the optimal policy. We measure this "agreement" by taking the sum of the probabilities of the optimal action as given by the policy we are evaluating over all states. We can see in figure 1a that the policy learned by SAIL agrees with the optimal policy in more states than the supervised policy. However, we can also see that there is still a larger number of states for which the policy hasn't converged to the optimal policy yet. This is because the majority of states are hit very infrequently and the agent usually only needs to learn the correct action for a small number of states to recover.

5 Conclusion

In this work we showed a way for an agent to reproduce states in the setting of imitation learning and show that this can help the agent reproduce demonstrations more accurately than pure supervised learning. In the future, we would like to evaluate our approach in more complex state spaces using nonlinear function approximators. Second, note that in the current formulation temporal difference learning is used to learn a function that has as many outputs as the policy has parameters. We would like to approximate this function in a more compact way to address scalability when using function approximators that have a high number of parameters. Approaches that learn gradients for deep neural networks with many parameters exist (including, but not limited to synthetic gradients [3]) but typically make assumptions based on the nature of the loss function that do not apply to our objective function. We also aim to improve the convergence rate by using off-policy updates and to compare the resulting algorithm more directly to IRL methods. We believe that this approach will be a flexible and powerful alternative to existing imitation learning approaches.

References

[1] Abdeslam Boularias, Jens Kober, and Jan Peters. "Relative Entropy Inverse Reinforcement Learning". In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)* 15 (2011), pp. 1–8.

[2] Sonia Chernova and Andrea L Thomaz. "Robot learning from human teachers". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8.3 (2014), pp. 1–121.

[3] Max Jaderberg et al. "Decoupled Neural Interfaces using Synthetic Gradients". In: *arXiv preprint* (2016), 1608.05343v1 [cs.LG]. arXiv: `1608.05343`.

[4] Edouard Klein et al. "A cascaded supervised learning approach to inverse reinforcement learning". In: *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)* (2013), pp. 1–16.

[5] Edouard Klein et al. "Inverse Reinforcement Learning through Structured Classification". In: *2Advances in Neural Information Processing Systems* (2012), pp. 1–9.

[6] Tetsuro Morimura et al. "Derivatives of logarithmic stationary distributions for policy gradient reinforcement learning". In: *Neural computation* 22.2 (2010), pp. 342–376.

[7] Dean a Pomerleau. "Alvinn: An autonomous land vehicle in a neural network". In: *Advances in Neural Information Processing Systems 1* (1988), pp. 305–313.

[8] Stéphane Ross and J Andrew Bagnell. "Efficient Reductions for Imitation Learning". In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)* 9 (2010), pp. 661–668.

[9] Stéphane Ross, Geoffrey Gordon, and J. Andrew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)* 15 (2011), pp. 627–635. arXiv: `arXiv:1011.0686v3`.

[10] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[11] John N Tsitsiklis and Benjamin Van Roy. "Average cost temporal-difference learning". In: *Automatica* 35 (1999), pp. 1799–1808.

[12] Brian D Ziebart et al. "Maximum Entropy Inverse Reinforcement Learning". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2007, pp. 1433–1438.

A confidence-based reinforcement learning model for perceptual learning

Matthias Guggenmos
Department of Psychiatry and Psychotherapy
Charité Universitätsmedizin Berlin
Charitéplatz 1, 10117 Berlin
matthias.guggenmos@charite.de

Philipp Sterzer
Department of Psychiatry and Psychotherapy
Charité Universitätsmedizin Berlin
Charitéplatz 1, 10117 Berlin
philipp.sterzer@charite.de

Abstract

It is well established that learning can occur without external feedback, yet normative reinforcement learning theories have difficulties explaining such instances of learning. Recently, we reported on a confidence-based reinforcement learning model for the model case of perceptual learning (Guggenmos, Wilbertz, Hebart, & Sterzer, 2016), according to which the brain capitalizes on internal monitoring processes when no external feedback is available. In the model, internal confidence prediction errors – the difference between current confidence and expected confidence – serve as teaching signals to guide learning. In the present paper, we explore an extension to this idea. The main idea is that the neural information processing pathways activated for a given sensory stimulus are subject to fluctuations, where some pathway configurations lead to higher confidence than others. Confidence prediction errors strengthen pathway configurations for which fluctuations lead to above-average confidence and weaken those that are associated with below-average confidence. We show through simulation that the model is capable of self-reinforced perceptual learning and can benefit from exploratory network fluctuations. In addition, by simulating different model parameters, we show that the ideal confidence-based learner should (i) exhibit high degrees of network fluctuation in the initial phase of learning, but reduced fluctuations as learning progresses, (ii) have a high learning rate for network updates for immediate performance gains, but a low learning rate for long-term maximum performance, and (iii) be neither too under- nor too overconfident. In sum, we present a model in which confidence prediction errors strengthen favorable network fluctuations and enable learning in the absence of external feedback. The model can be readily applied to data of real-world perceptual tasks in which observers provided both choice and confidence reports.

Keywords: Reinforcement learning, perceptual learning, confidence, feedback, exploration

Acknowledgements

This work was supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG, FOR 1617: grants STE1430/6-1 and STE1430/6-2)

1 Introduction

A well-established form of learning without external feedback is perceptual learning – the ability to improve sensory information processing through training or repeated exposure. Previous models of perceptual learning have successfully used principles from reinforcement learning to explain perceptual learning, but they were limited to cases of external feedback (Kahnt, Grueschow, Speck, & Haynes, 2011; Law & Gold, 2008). Recently, we introduced a novel perceptual learning model, in which learning was guided by internal confidence-based reinforcement signals (Guggenmos, Wilbertz, Hebart, & Sterzer, 2016). Using an associative reinforcement learning rule – a combination of Hebbian learning and confidence prediction errors – the model tuned network weights to improve perception. The model successfully explained perceptual learning behavior in human participants and found dopaminergic source and target regions as neural correlates of confidence-based learning signals, further corroborating a reinforcement learning account of perceptual learning in the absence of external feedback.

However, a simplifying assumption of this previous model was that the variability of confidence reports given constant stimuli was modeled as noise. Here we offer a different treatment of this aspect with the key idea that variability in confidence reports is directly related to network fluctuations in the processing of a stimulus. More specifically, we propose that sensory processing networks employ slightly different pathways configurations for repeated presentations of the same stimulus, where some configurations lead to higher perceptual confidence than others. In the brain, such fluctuations may arise either due to the probabilistic nature of neural information processing or due to an explicit exploratory capacity of neural circuits. Learning in our model is guided by confidence prediction errors that serve to strengthen pathway configurations for which fluctuations led to above-average confidence and to weaken those associated with below-average confidence outcomes. The aim of this work is to introduce the formulation of such a model, to assess its learning behavior and to investigate the influence of key model parameters.

2 Model

We consider a classic perceptual learning task, in which observers have to identify whether a noisy Gabor stimulus is oriented counterclockwise (ccw) or clockwise (cw) with respect to a reference axis. The model consists of two sensory input nodes, which represent energy detectors E_{ccw} and E_{cw} and two decisional output nodes A_{ccw} and A_{cw}. These input and output layers are fully connected with signal weights $w_{\mathrm{ccw,ccw}}$ and $w_{\mathrm{cw,cw}}$ connecting input and output nodes with same orientations and noise weights $w_{\mathrm{ccw,cw}}$ and $w_{\mathrm{cw,ccw}}$ connecting nodes of opposing orientations. Importantly, the specific processing of a given stimulus is subject to trial-to-trial fluctuations. Fluctuations are implemented by means of a trial-dependent multiplicative (balancing) factor b. For instance, the output activities A_{ccw} and A_{cw} for a counterclockwise stimulus are:

$$A_{\mathrm{ccw}} = E_{\mathrm{ccw}} b w_{\mathrm{ccw,ccw}} + E_{\mathrm{cw}} w_{\mathrm{cw,ccw}} \tag{1}$$

$$A_{\mathrm{cw}} = E_{\mathrm{cw}} w_{\mathrm{cw,cw}} + E_{\mathrm{ccw}} (2-b) w_{\mathrm{ccw,cw}} \tag{2}$$

The factor b thus shifts the balancing between counterclockwise stimulus energy being processed via the signal pathway ($w_{\mathrm{ccw,ccw}}$) or the noise pathway ($w_{\mathrm{ccw,cw}}$).

Confidence c is computed as the absolute value of the difference between A_{ccw} and A_{cw}:

$$c = \lambda |A_{\mathrm{ccw}} - A_{\mathrm{cw}}| \tag{3}$$

where λ is the confidence sensitivity parameter. Learning is guided by a confidence prediction error δ, defined as the difference between current confidence c and expected confidence $\overline{c}$:

$$\delta = c - \overline{c} \tag{4}$$

Expected confidence is estimated via a Rescorla-Wagner learning rule:

$$\overline{c} \leftarrow \overline{c} + \alpha_c \delta \tag{5}$$

Finally, weights are updated by means of an associative learning reinforcement learning rule:

$$w_{\mathrm{ij}} \leftarrow w_{\mathrm{ij}} + \alpha_w E_{\mathrm{i}} A_{\mathrm{j}} \delta \tag{6}$$

To prevent unlimited growth of the weights, synaptic normalisation is performed:

$$w_{\mathrm{ij}} \leftarrow \frac{w_{\mathrm{ij}}}{\frac{1}{2}(w_{\mathrm{ij}} + w_{\mathrm{jj}})} \tag{7}$$

Choices are based on a softmax action selection rule:

$$p(\mathrm{ccw}) = 1 + \frac{1}{1 + e^{-\beta(A_{\mathrm{ccw}} - A_{\mathrm{cw}})}} \tag{8}$$

A key feature for empirical application (e.g., model fitting to behavioral data) is that the trial-by-trial balancing factor b can be computed from confidence reports by solving Equation (3) for b. For counterclockwise stimulus presentation, the analytic derivation for b is as follows:

$$b = \frac{2E_{\mathrm{ccw}} w_{\mathrm{ccw,cw}} + E_{\mathrm{cw}}(w_{\mathrm{cw,cw}} - w_{\mathrm{cw,ccw}}) \pm \frac{c}{\lambda}}{E_{\mathrm{ccw}}(w_{\mathrm{ccw,cw}} + w_{\mathrm{ccw,ccw}})} \tag{9}$$

Caused by the absolute value in Equation (3), a single ambiguity remains for b in the term $\pm c/\lambda$. As a heuristic, we recommend to choose whichever sign produces a b with the smallest deviation from 1, thus slightly biasing the model towards a smaller influence of the balancing component.

3 Simulation results

To assess the behavior of the model and its dependence on parameters, we repeatedly confronted the model with 5000 trials of clockwise or counterclockwise stimulus presentations. Each parameter combination was simulated for 10000 runs. Stimuli were a mixture of the designated and the opposing stimulus orientation with a ratio of 3:1. To make the orientation discrimination task challenging, white Gaussian noise was added. The balancing factor b was drawn from the following uniform distribution:

$$b \sim [1 - \frac{v_b}{2}; 1 + \frac{v_b}{2}] \tag{10}$$

where v_b determined the magnitude of trial-to-trial weight fluctuations.

For all simulations, if not otherwise specified, the following parameters were chosen: $v_b = 0.5$, $= 1$, $\alpha_w = 0.2$, $\alpha_c = 0.2$, $\beta = 3$. Signal weights were initialized with 1.25, noise weights with 0.75, and initial expected confidence was set to 0.

3.1 Successful perceptual learning in the absence of external feedback

Figure 1 shows the the behavior of the model with the above default parameters. Within the first 2000 trials, signal weights have increased and noise weights have decreased to their respective saturation level. As expected from such a tuning of the network, the proportion of correct responses increased as a consequence (from around 0.59 to 0.78). Since confidence was computed from the output activities of the model, confidence and expected confidence likewise showed an increase during learning. Overall, this simulation demonstrated that the model was capable of learning, achieved by an increase of signal weights and a decrease of noise weights.

3.2 Model parameters affect learning speed and saturation performance

Three parameters directly influenced learning: the magnitude of weight fluctuations (determined by v_b), the weight-related learning rate α_w and the confidence sensitivity parameter λ, which represented the under/over-confidence of the observer. In the following, we discuss the learning behavior of the model in dependence of each of these parameters.

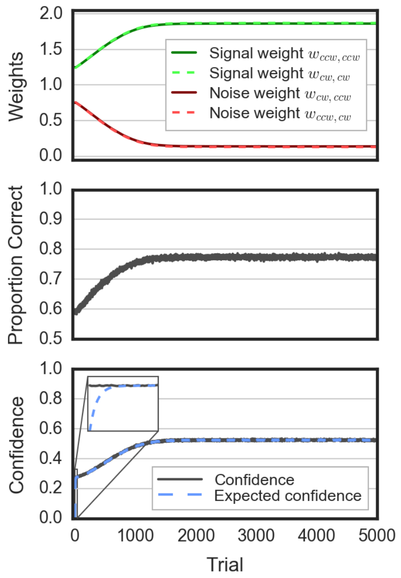


Figure 1: Behavior of the confidence-based perceptual learning model. Learning is achieved by an increase of signal weights and a decrease of noise weights. As a consequence of the weight tuning, performance (proportion correct responses), confidence and expected confidence increase until saturation is reached.

3.2.1 Magnitude of weight fluctuations

The parameter v_b was the degree to which the network explored different weight configurations. In the initial phase of learning (trials 1-1000), learning was fastest for intermediate values ($v_b \sim 1$) (Figure 2A). Too low values of v_b led to slower learning as the exploration of weight configurations was negligible and thus had no noticeable positive effect. Too high values of v_b often induced counterproductive weight configurations, erroneously leading to high confidence for wrong choices. While there was thus a certain sweet spot of v_b for the initial phase of learning, saturation performance generally decreased with higher v_b, as the explorativeness became a disadvantage once the network had learned.

3.2.2 Weight-related learning rate

The parameter α_w determined how much the network learned from confidence prediction errors. In the initial phase of learning, higher learning rates generally led to faster learning (Figure 2B). However, in the long run high learning rates led to suboptimal saturation performance, implying a long-term advantage for slow but steady learners.

3.2.3 Confidence sensitivity

The parameter λ determined the degree of confidence for a given activation difference between counterclockwise and clockwise output activities. In the initial phase of learning, fastest learning was achieved for intermediate values ($\lambda \sim 3$) (Figure 2C). Too high values of λ implied overconfidence and led to overhasty, often disadvantageous, updates of the network. Too low values of λ led to tiny confidence-based learning signals and thus reduced learning. Yet, long-term performance levels were almost unaffected by λ.

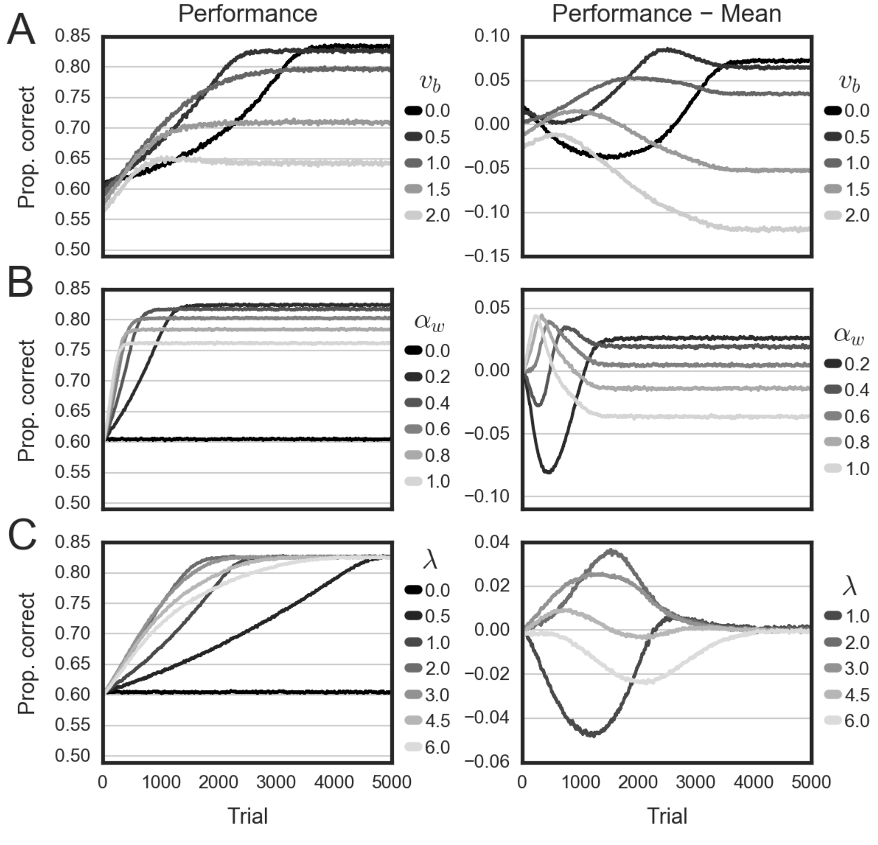


Figure 2: Performance in dependence of model parameters. Left panels show the proportion of correct responses across trials for different parameter settings. In right panels, the mean across parameters was subtracted, for better illustration.
(A) Magnitude of weight fluctuations v_b.
(B) Weight-related learning rate α_w.
(C) Confidence sensitivity λ.

4 Conclusions

Here we introduced a novel computational model for perceptual learning in the absence of external feedback, utilizing network fluctuation in combination with confidence prediction errors.

We showed through simulation that the model was capable of learning by increasing signal weights and decreasing noise weights. Importantly, we found that the exploratory network component, implemented as trial-to-trial fluctuations of the weight matrix, was able to boost perceptual learning in the initial phase of learning. However, once the network has reached a stable level, large network fluctuations became a disadvantage and an ideal observer should thus decrease network exploration. This pattern of results mirrors the classic exploration-exploitation dilemma, in which ideal agents have to balance exploration and exploitation depending on their outcomes and the dynamics of the environment.

We further showed that ideal confidence-based learners should neither be too underconfident nor too overconfident for optimal initial learning, while under/overconfidence had little effect on long-term performance levels. By contrast, for the weight-related learning rate, initial gains in performance due to high learning rates came at the cost of reduced long-term performance.

In the future, we plan to apply the model to real-world perceptual learning datasets of human participants in order (i) to assess the model evidence in comparison to prior models, and to (ii) investigate potential individual differences related to the degree of network fluctuations, and how those relate to learning.

References

Guggenmos, M., Wilbertz, G., Hebart, M. N., & Sterzer, P. (2016). Mesolimbic confidence signals guide perceptual learning in the absence of external feedback. eLife, 5.

Kahnt, T., Grueschow, M., Speck, O., & Haynes, J.-D. (2011). Perceptual learning and decision-making in human medial frontal cortex. Neuron, 70(3), 549559.

Law, C.-T., & Gold, J. I. (2008). Neural correlates of perceptual learning in a sensory-motor, but not a sensory, cortical area. Nature Neuroscience, 11(4), 505513.

What is the nature of decision noise in random exploration?

Siyu Wang
Department of Psychology
University of Arizona
Tucson, AZ 85721
sywangr@email.arizona.edu

Robert C. Wilson
Department of Psychology
University of Arizona
Tucson, AZ 85721
bob@arizona.edu

Abstract

The explore-exploit tradeoff is a fundamental behavioral dilemma faced by all adaptive organisms. Should we explore new options in the hopes of finding a better meal, a better house or a better mate, or should we exploit the options we currently believe to be best? Striking the right balance between exploration and exploitation is hard computational problem and there is significant interest in how humans and other animals make explore-exploit decisions in practice. One particularly effective strategy for solving the explore-exploit dilemma is choice randomization [1]. In this strategy, the decision process is corrupted by noise meaning that high value "exploit" options are not always chosen and exploratory choices are sometimes made by chance. In theory, such "random exploration", can be surprisingly effective in explore-exploit problems and, if implemented correctly, can come close to optimal performance. Recent work suggests that humans actually use random exploration to solve simple explore-exploit problems [2]. Despite this progress a number of questions remain about the nature of random exploration as there are a number of ways in which seemingly stochastic choices could be generated. In one strategy, that we call the "external noise strategy", participants could rely on stochasticity in the world and allow irrelevant features of the stimulus to drive choice. In another strategy called "internal noise strategy", people could rely on stochastic processes within their own brains. In this work, we modified our recently published "Horizon Task" in such a way as to distinguish these two strategies. Using both a model-free and model-based analysis of human behavior we show that, while both types of noise are present in explore-exploit decisions, random exploration is dominated by internal noise. This suggests that random exploration depends on adaptive noise processes in the brain which are subject to (perhaps unconscious) cognitive control.

Keywords: explore-exploit, decision noise, random exploration, hierarchical Bayesian analysis

Acknowledgements:

We'd like to thank all the research assistants who run the experiment: Maxwell Reece Alberhasky ,Chrysta Andrade ,Daniel Carrera ,Kathryn Chung ,Emily Giron ,Brittney Gonzalez ,Leah Hall ,Maura Higgs ,Min-Hee Kang ,Kathryn Lui Kellohen ,Hannah Marie Kyllo ,Alex Lawwill ,Michael de Leon ,Stephanie Low ,Colin Lynch ,Genevieve Patterson ,Filipa Santos ,Shlishaa Savita ,Catie Sikora ,Vera Thornton ,Guillermo Vargas ,Christina West.

1 Introduction

The explore-exploit tradeoff is a fundamental behavioral dilemma faced by all adaptive organisms. Should we explore new options in the hopes of finding a better meal, a better house or a better mate, or should we exploit the options we currently believe to be best? Striking the right balance between exploration and exploitation is hard computational problem and there is significant interest in how humans and other animals make explore-exploit decisions in practice. One particularly effective strategy for solving the explore-exploit dilemma is choice randomization [1]. In this strategy, the decision process between exploration and exploitation is corrupted by "decision noise", meaning that high value "exploit" options are not always chosen and exploratory choices are sometimes made by chance. In theory, such "random exploration", is surprisingly effective and, if implemented correctly, can come close to optimal performance [10].

Recently we showed that humans actually use random exploration and that they actively adapt their decision noise to solve simple explore-exploit problems in the lab [2]. However, a key limitation of this work was that it was that it was that the source of the decision noise used for exploration is unknown. Of particular interest is whether the adaptive decision noise that is linked to exploration is generated internally, within the brain, or arises externally, in the input from the world. Previous work makes a strong case for both types of noise being relevant to behavior. For instance, external, stimulus-driven noise is thought to be a much greater source of choice variability in perceptual decisions than internal noise [3]. Conversely internal, neural noise is thought to drive exploratory singing behavior in song birds [4] and the generation and control of this internal noise has been linked to specific neural structures.

In this paper, we investigate which source of noise, internal vs external, drives random exploration in humans in a simple explore-exploit task adapted from our previous work [2]. To distinguish between the two types of noise, we had people make the exact same explore-exploit decision twice. If decision noise is purely externally driven, then people choices should be identical both times, that is their choices should be consistent since the stimulus is the same both times. Meanwhile, if noise is internally driven, the extent to which their choices are consistent should be determined by the level of the internal noise. By analyzing behavior on this task in both a model-free and model-based way, we were able to show that, while both types of noise are present in explore-exploit decisions, the contribution of internal noise to random exploration far exceeds that contributed by the stimulus.

2 Methods

2.1 Participants

78 Participants (37 male, 41 female, ages 17-25) from the UA subject pool, played the task. 27 were excluded on the basis of performance (using the same exclusion criterion as in [2]) leaving 51 for the analysis.

2.2 Behavioral task

Participants played a series of games of a sequential two-armed bandit gambling task. In each game they made repeated decisions between two options. Each option paid out a random reward between 0 and 100 points sampled from a Gaussian distribution with a fixed standard deviation of 8 points. The means of the underlying Gaussian were different for the two bandit options, remained the same within a game, but changed with each new game. Participants were instructed to maximize the points earned over the entire task. The number of games participants play depends on how well they perform, the better they perform, the sooner the task will end. On average, participants play 128 games.

The first four trials of each game were forced-choice trials, in which only one of the options was available for the participant to choose. We used these forced-choice trials to manipulate the relative ambiguity of the two options, by providing the participant with different amounts of information about each bandit before their first free choice. The four forced-choice trials set up two uncertainty conditions: unequal uncertainty(or [1 3]) in which one option was forced to be played once and the other three times, and equal uncertainty(or [2 2]) in which each option was forced to be played twice. After the forced-choice trials, participants made either 1 or 6 free choices (two horizon conditions).

The crucial additional manipulation in this task is repeated games. Each of the set of forced-choice trials appeared twice in the game. For each pair of repeated games, they share the exact same set of forced-choice trials. But for free-choice trials, each game in the repeated pair sampled independently from the underlying Gaussian distribution using the same underlying set of parameters (the mean of the Gaussian for both bandits). In order not to let the participants notice that games are repeated, we randomize the order of games so that each repeated pair is at least five games away from each other.

2.2 Model-free analysis

As in our previous paper [2], the influence of information bias is measured as the probability of choosing the more informative option, i.e. the bandit with only 1 outcome in [1 3] condition; decision noise is measured as the probability of choosing the low mean option in [2 2] condition.

In addition, whether participants make consistent choices in repeated games is used as a model-free measure of internal noise, since in repeated trials, external noise should be identical on both trials. So only internal noise can differ and drive the choice inconsistency. The degree to which people make consistent choices in repeated trials can reflect the internal noise.

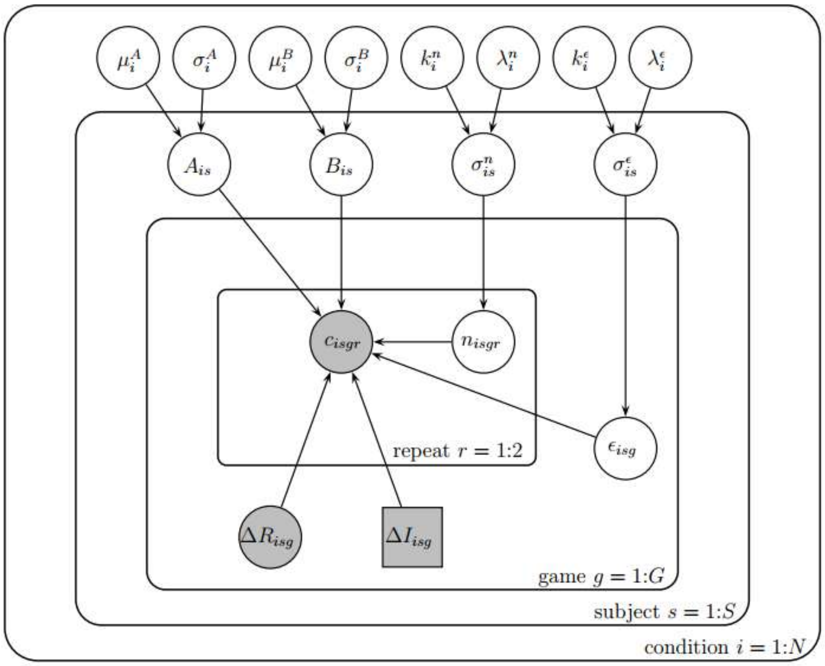
2.3 Model-based analysis

As with our model-free analysis, the model-based analysis focuses only on the first free-choice trial since that is the only free choice when we have control over the information bias between the two bandits. To model participants' choices on this first free-choice trial, we assume that they make decisions by computing the difference of value ΔQ between the two options:

$$\Delta Q = \Delta R + A\,\Delta I + b + \epsilon + n$$

Where, the experimentally controlled variables are $\Delta R = R_{right} - R_{left}$, difference between the mean of rewards shown on the forced trials, and ΔI, the difference information available for playing the two options on the first free-choice trial (for simplicity we define ΔI to be +1, -1 or 0: +1 if one reward is drawn from the right option and three are drawn from the left in the [1 3] condition, -1 if one from the left and three from the right, and in [2 2] condition, ΔI is 0).

The subject-and-condition-specific parameters are: the spatial bias, b, which determines the extent to which participants prefer the option on the right, the information bonus A, which controls the level of directed exploration, ϵ denotes the external, external noise, which is identical on the repeat versions of each game, and n denotes internal noise, which is uncorrelated between repeat plays. For repeated games, the set of



Priors

$\mu_i^A \sim \text{Gaussian}(0, 100)$, $\sigma_i^A \sim \text{Exponential}(0.1)$

$\mu_i^B \sim \text{Gaussian}(0, 100)$, $\sigma_i^B \sim \text{Exponential}(0.1)$

$k_i^n \sim \text{Exponential}(0.1)$, $\lambda_i^n \sim \text{Exponential}(10)$

$k_i^\epsilon \sim \text{Exponential}(0.1)$, $\lambda_i^\epsilon \sim \text{Exponential}(10)$

Subject specific parameters

$A_{is} \sim \text{Gaussian}(\mu_i^A, \sigma_i^A)$

$B_{is} \sim \text{Gaussian}(\mu_i^B, \sigma_i^B)$

$\sigma_{is}^n \sim \text{Gamma}(k_i^n, \lambda_i^n)$

$\sigma_{is}^\epsilon \sim \text{Gamma}(k_i^\epsilon, \lambda_i^\epsilon)$

Stimulus driven noise for repeated game

$\epsilon_{isg} \sim \text{Gaussian}(0, \sigma_{is}^\epsilon)$

Free noise for each game

$n_{isgr} \sim \text{Gaussian}(0, \sigma_{is}^n)$

Observed choices

$\Delta Q_{isgr} \leftarrow \Delta R_{isg} + A_{is}\Delta I_{isg} + B_{is} + n_{isgr} + \epsilon_{isg}$

$c_{isgr} \sim \text{Bernoulli}\left(Q_{isgr} > 0\right)$

FIGURE 1 – Hierarchical Bayesian model

forced-choice trials are exact the same, so the external noise should be the same, but the free decision noise may be different. This is exactly how we distinguish external noise from internal noise. In symbolic terms, for repeated games i and j, $\epsilon_i = \epsilon_j \; and \; n_i \neq n_j$.

The decision model assumes that participants choose right if $\Delta Q > 0$, otherwise the participant will choose the left option.

We used hierarchical Bayesian analysis to fit the parameters (see Figure 1 for an graphical representation of the model in the style of [5]). In particular, we fit values of the information bonus A, spatial bias B, internal noise n variance, σ^n and external noise ε variance, σ^ε for each participant in each horizon and uncertainty condition. The mean and standard deviation of information bonus A and spatial bias B are sampled from a Gaussian prior and an exponential prior respectively. The variance for both type of noises were sampled from a gamma distribution, and the group-level parameter k and λ for the gamma distribution are sampled from exponential priors. At the game level, we sampled one stimulus driven noise for each pair of repeated games and we sampled two free decision noises for each individual game in the pair. The model in Figure 1 is fitted using the MATJAGS and JAGS [6].

3 Results

We show that both internal and external noise contribute to the behavioral variability in random exploration. Using hierarchical Bayesian analysis, we also show that the effect of internal noise is the main source of noise in random exploration.

3.1 Model free analysis demonstrates the existence of internal and external noise.

In line with previous findings, both directed and random exploration increase with horizon (See Figure 2. A, B). In addition, choice inconsistency between repeated games was non-zero in both horizon conditions, suggesting that not all of the noise was stimulus driven. In addition, choice inconsistency was higher in horizon 6 than in horizon 1 for both [1 3] and [2 2] condition (See Figure 2.C), suggesting that at least some of the horizon dependent noise is internal.

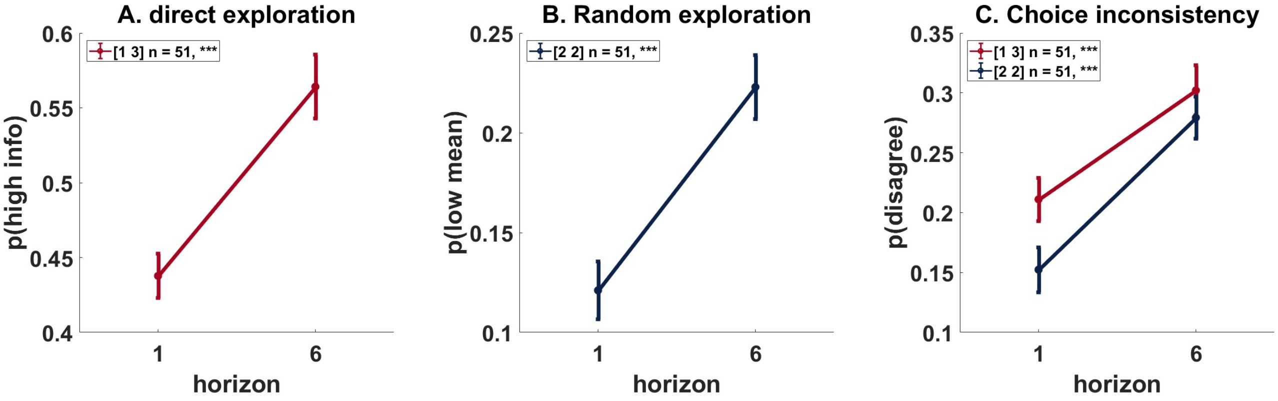


FIGURE 2 – Both direct (P < 0.001) and random exploration (P < 0.001) increases with horizon. Choice consistency in both [1 3] and [2 2] condition increases with horizon($P_{[13]} < 0.001$, $P_{[22]}<0.001$).

3.2 Model-based analysis shows that internal noise is the main source of noise in exploration

To more precisely quantify the size of internal and external noise in this task, we turned to model fitting. Both external(P<0.001) and internal noise (P<0.001) increase with horizon (See Figure 3). In both horizon 1 and horizon 6, internal noises are significantly higher than internal noises (for both horizon 1 and horizon 6, P < 0.001).

Figure 3 also demonstrates the existence of both external noise and internal noise. Hierarchical Bayesian analysis provides us a way to quantify the magnitude of external and internal noise respectively, from the result, internal noise is the main source of random exploration.

4 Discussion

In this paper, we investigated whether random exploration is driven by internal noise, putatively arising in the brain, or external noise, arising from the environment. We find horizon dependent changes in both internal and external sources of noise, but that the effect of internal noise is much greater.

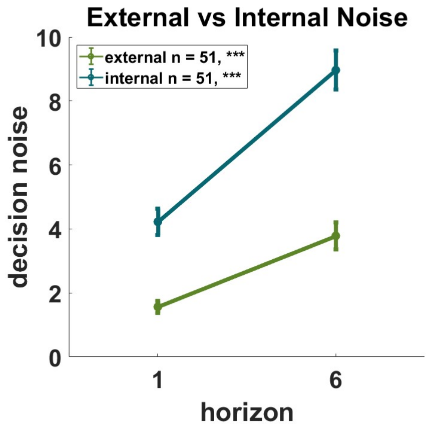


FIGURE 3 – Both external and internal noise increase with horizon. But internal noise has a higher magnitude and is the main source of noise in random exploration.

One limitation of this work is in the interpretation of the different noises as being internal and external. In particular, while we controlled many aspects of the stimulus across repeated games (e.g. the outcomes and the order of the forced trials), we could not perfectly control *all* stimuli the participant received which, for example, would vary based on where they were looking. Thus, our estimate of external noise is likely a lower bound. Likewise, our estimate of internal noise is likely an upper bound as these "missing" sources of stimulus driven noise would be interpreted as internal noise in our model. Despite this, the sheer magnitude of the difference between internal and external noise (internal noise is 2-3 times the size of external noise, Figure 3), suggests that our interpretation may be safe as an awful lot of external noise would have to be explained by variables not relevant to the task.

The horizon-dependent increase in internal noise is consistent with the idea that random exploration is driven by intrinsic variability in the brain. This is in line with work in the bird song literature in which song variability during song learning has been tied to neural variability arising from specific areas of the brain. In addition, this work is consistent with a recent report from Ebitz et al [7] in which behavioral variability in an "explore" state was also tied to internal rather than external sources of noise. Whether such a noise-controlling area exists in the human brain is less well established, but one candidate theory [4] suggests that norepinephrine (NE) from the locus coeruleus may play a role in modulating internal levels of noise. While there is some evidence that NE plays a role in explore-exploit behavior [8], this link has been questioned [9].

References

[1] WR. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika, 25(3-4):285–294, 1933.

[2] RC. Wilson, A. Geana, JM. White, EA. Ludvig & JD. Cohen, Humans use directed and random exploration to solve the explore-exploit dilemma, Journal of Experimental Psychology: General, 143(6) 2074, 2014.

[3] BW. Brunton, MM. Botvinick, CD. Brody. Rats and humans can optimally accumulate evidence for decision-making. Science, 340, 95-98, 2013.

[4] G. Aston-Jones, JD. Cohen. An integrative theory of locus coeruleus norepinephrine function: adaptive gain and optimal performance. Annual Review of Neuroscience, 28, 403-450, 2005.

[5] MD. Lee, E.-J. Wagenmakers. Bayesian Modeling for Cognitive Science: A Practical Course. Cambridge University Press. 2013

[6] Martyn Plummer. JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling, Proceedings of the 3rd International Workshop on Distributed Statistical Computing, 2003.

[7] B. Ebitz ,T. Moore & T. Buschman, Bottom-up salience drives choice during exploration, Cosyne, 2017.

[8] J. Marieke. Neural Mechanisms Underlying the Induction and Relief of Perceptual Curiosity. Frontiers in Behavioral Neuroscience, 2017.

[9] Nieuwenhuis S., Heslenfeld D.J., von Geusau N.J., Mars R.B., Holroyd C.B., Yeung N. Activity in human reward-sensitive brain areas is strongly context dependent. NeuroImage.25:1302–1309, 2005.

[10] Bridle, J. S. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimates of parameters. D. S. Touretzky (Ed.), Advances in Neural Information Processing Systems, 2, 211–217, 1990.

Effective, Time-Efficient State Representations for Human-Machine Decision Making

Jaden B. Travnik
Department of Computing Science
University of Alberta
Edmonton, AB, Canada, T6G 2E1
travnik@ualberta.ca

Patrick M. Pilarski
Department of Medicine &
Department of Computing Science
University of Alberta
Edmonton, AB, Canada, T6G 2E1
pilarski@ualberta.ca

Abstract

Humans and learning machines need to make decisions together in increasingly complex environments. The way the world is presented to a machine learner, termed its *representation* of the world, is a critical part of this shared decision-making process. One representative case is that of robotic prostheses attached to the human body to help mitigate the loss of limb function due to amputation, injury, or illness. With advances in functionality and available sensor data, prosthetic devices are now faced with a growing stream of real-time sensorimotor data that they must interpret to execute the motor intentions of their human users. This decision making must be rapid, accurate, and performed with the limited computational resources of a wearable robot. Reinforcement learning is therefore a natural choice to learn predictions and policies that map situations to human intent and control outcomes. Critical to all existing approaches for reinforcement learning in prosthesis control is the time efficient and effective representation of an increasing sensorimotor data stream to a reinforcement learning control system. In this study, we present a first look into the impact of increased prosthetic sensorimotor information on the computation time and prediction performance of two different linear representations: the widely used tile coding and a novel modification of Kanerva coding, *selective Kanerva coding*. Our results not only confirm potential limitations to tile coding as the number of sensory input dimensions increase, but also show evidence that selective Kanerva coding can provide an effective representation within a limited computational time budget. This is, in part, possible because the dimensionality of the state space does not affect the size of the representation selective Kanerva coding provides. This study therefore takes a first step towards efficiently representing complex, high-dimensional data streams to machine learning control systems.

Keywords: Reinforcement Learning, Sates Representations, Human-Machine Interaction, Real-time machine learning

Acknowledgements

This research was undertaken, in part, thanks to funding from the Canada Research Chairs program, the Canada Foundation for Innovation, and was supported by the Alberta Machine Intelligence Institute (Amii), Alberta Innovates – Technology Futures (AITF), the Government of Alberta, and the Natural Sciences and Engineering Research Council (NSERC).

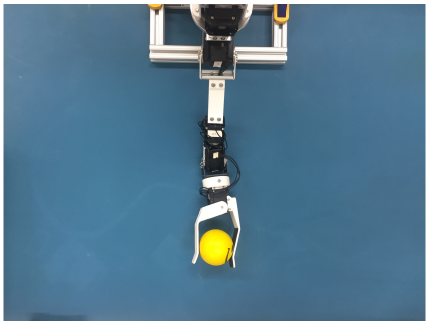


Figure 1: The upper-limb research prosthesis used in this study (the Bento Arm [3]).

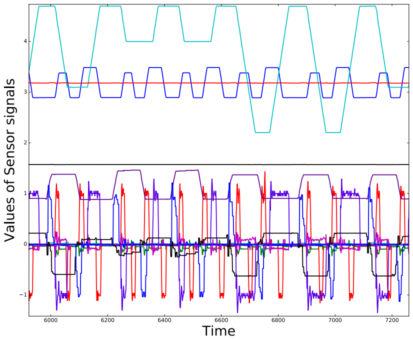


Figure 2: Example of the position, velocity, load, and temperature sensor signals generated during its operation on a simple repetitive task.

1 INTRODUCTION

Prosthetic limbs and other assistive rehabilitation technologies rely on their sensors to respond appropriately to the intentions and needs of their human users. The use of machine learning approaches such as pattern recognition allow available sensors to be further leveraged to increase the number of functions controllable by a user [2]. There is now convincing evidence that machine learning control approaches such as pattern recognition can enable patients with amputations to sequentially control a device with a robotic elbow, wrist, and hand with multiple discrete grasp patterns—far more degrees of control than were previously possible with conventional control solutions [1, 2]. This increase in function can be attributed both to an increase in the number of sensors deployed within a prosthetic socket, and the efficient use and synthesis of the information provided by these sensors. The combination of sensorimotor signals into a useful summary of a system's state, termed a *state representation*, has become increasingly important to the performance of prosthetic devices, especially those that leverage pattern recognition, regression, or real-time machine learning [1]. If used carefully, these signals present a valuable window into the intent of a human user and their prosthesis's interactions with a changing, daily-life environment.

A prosthesis must be able to approximate and react to its sensor inputs within a short window of time in order to remain effective [4]. If a representation can be formed for a control system in an efficient way, even in the face of high-dimensional sensor data, it can readily be *computed, stored, and used in real time* on the computationally limited hardware embedded within wearable prosthetic devices.

Previous work by our group has explored in detail the use of real-time machine learning methods for prosthetic control—we have to date performed a wide range of prosthetic prediction and control studies wherein subjects with and without amputations used different techniques from the field of reinforcement learning (RL) to operate a robotic arm, e.g., [5–11]. In all of these studies we exclusively relied on the linear representation method known as tile-coding (c.f., [12]) to provide our RL prediction and control systems with a view into the available space of prosthesis- and user-derived sensorimotor signals. These linear representations were chosen because they are highly efficient in both computation and data usage for a fixed (small) number of input signals. However, as prosthetic devices improve and the number of sensors available to the system increases, it remains unclear how standard linear representations like tile-coding are affected, and if more scalable representations exist which would be more aptly suited for use in prosthetic limbs and other wearable assistive rehabilitation technologies.

Finding a good representation of input signals in complex, high-dimensional, and continuous environments is a difficult problem faced not just in the adaptive control of prosthetic limbs, but in a wide range domains where designers seek to deploy machine learning for real-world control tasks; notable examples range from speech recognition to self-driving vehicles. However, unlike enterprise scale applications with access to vast computational resources, the prosthetic setting requires that all computation be limited to run on small, power efficient processors that can be installed in a self-contained way within the chassis of a prosthetic device. The representation of high-dimensional continuous state information therefore needs to be done in a computationally efficient fashion.

In this work we therefore contribute a first study on the effects of increasing the dimensionality of prosthetic sensory data in terms of computation time and prediction accuracy of on policy general value functions (GVFs) using linear tile-coding representations (the dominant function approximation approach used in RL) and propose a novel modification of a competitor coding approach that promises to scale accurately and efficiently as the number of sensory dimensions increases on an assistive device [11, 13].

2 REPRESENTATION USING LINEAR METHODS

Approximating a value function using linear methods is considered a standard approach because of the ease of computing the gradient of the value function with respect to the learned parameters or weights [12]. This leads to a particularly simple form of the gradient-descent update with the guarantee that any convergence to a local optimum is a convergence to a global optimum. On top of their theoretical results, linear methods can excel in terms of computation and data efficiency but this depends critically on how the dimensions of states are represented in terms the representation's feature vector, ϕ.

2.1 Tile Coding

Tile coding (TC) is a linear representation that is often chosen in RL for it's efficiency in online learning. Tile coding splits up the d-dimensional state space into m overlapping and offset partitions called tilings and each tiling is split into a set of n tiles. Each tile represents a binary (0 or 1) feature that is activated when the state lies within the tile. A binary feature vector ϕ of length nm, can then be constructed by concatenating all n features for all m tilings. Since only m features are activated, tile coding has an advantage when choosing a constant step-size parameter, α, as the number of active features is independent of the state. For example, a reasonable step-size parameter might be $\alpha = 0.1/m$.

In order to capture the interaction between dimensions, tilings must span the state space so that each tile represents a small d-dimensional hyper-cube. This leads to exponentially many features ($dim(\phi) = m * n^d$). For small dimensions found in common control tasks, tile coding provides an easily implementable representation with clear benefits. However, as the dimension of the task grows, so does the memory cost.

2.2 Selective Kanerva Coding

Kanerva coding is the application of sparse distributed memory as a representation in reinforcement learning. Unlike tile coding, the number of features in Kanerva coding does not grow with the number of dimensions. This can lead to dramatic savings in necessary memory resources. However, a common issue when using Kanerva coding in RL is choosing an appropriate number of prototypes and activation radius and although several researchers have shown improved performance by moving the prototypes around the representation space or by growing the representation using experience from the learner, these methods add extra memory and computational complexity to the algorithm [15,16]. Additionally, as other work has shown that Kanerva coding may successfully be applied to continuous domains, the computational cost of high-dimensional continuous states has not yet been explored [17].

We propose *Selective Kanerva Coding* (SKC) to address these issues. The idea is to remove the activation radius and simply find the c closest prototypes to the state of the system using Hoare's quickselect which can be used to find the c shortest distances in an array of distances [18]. One way of choosing a good c is to choose a small ratio, η such that $c = \lfloor K\eta \rfloor$. Not only does this method still have the same $O(K)$ complexity as computing how many prototypes are within an activation radius, but it shares with tile coding the guarantee of the same number of activated features along with the associated benefits, e.g., in selecting learning step sizes. Utilized alongside True Online TD(λ), SKC promises to be an efficient, reliable representation for computing GVFs [14].

3 EXPERIMENT

A robotic arm designed to be worn as a research prosthesis (the Bento Arm of Dawson et al., [3], Fig. 1) was controlled in a sorting exercise where three different objects were picked up from the same location, carried, and then dropped at different locations assigned by their weight.The goal of the prediction task was then to predict what angle the shoulder angle sensor would report over the next 10 timesteps (∼0.3 seconds into the future). Predictions were made through the use of an on-policy general value function learned via true online temporal-difference learning. In order to predict 10 timesteps into the future, $\gamma = 0.9$ was used in specifying the GVF. Learning rate and the eligibility trace parameters were empirically set to $\alpha = 0.001$ and $\lambda = 0.99$, respectively.

To examine the effect of the number of sensor dimensions on prediction performance and computation time, three different sensor combinations were used—representations were constructed with 2, 8, and 12 sensory input signals. In the 2-dimensional setting, only the elbow angle and load were given as part of the representation, whereas in the 8-dimensional setting, the angle, load, temperature, and velocity of both the elbow and the wrist rotator were given. Finally, the angle, load, temperature, and velocity of the elbow, the wrist rotator, and the wrist flexor were given in the 12-dimensional task. Exactly 248 different combinations of tiles and tilings were used to generate a wide range of tile coding configurations and thus a wide range of features for each dimensional setting, ranging from 128 to 331776 features. One reasonable method of offsetting each tiling is to deterministically shift each tiling using a knight's-move

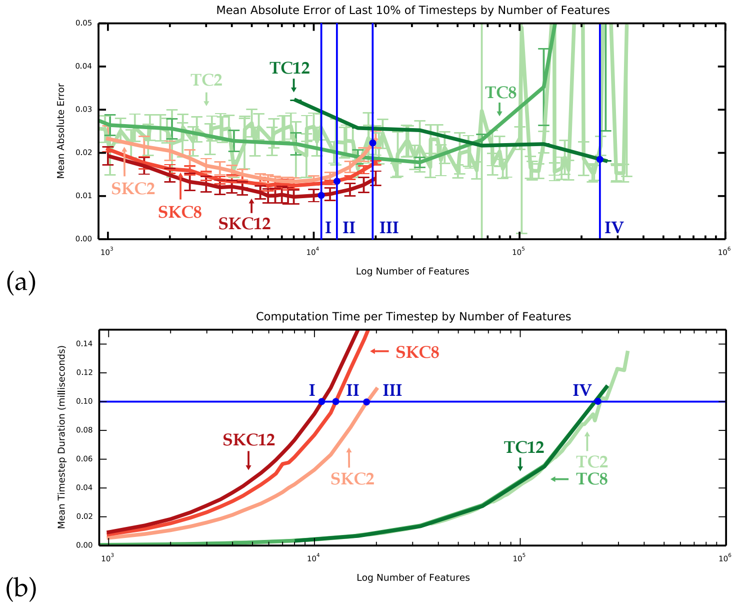


Figure 3: Key result: comparison of prediction error and computation time for tile coding and SKC as their number of features increases, including (a) the mean absolute prediction error on the last 10% of the combined data for SKC and tile coding on 2, 8, and 12 dimensions; and (b) the mean computation time of the combined data for SKC and tile coding on 2, 8, and 12 dimensions. The maximum features computable in a 100ms timestep is shown in both plots by blue lines and the points I, II, II, and IV for SKC12, SKC8, SKC2, and TC2/8/12, respectively.

pattern.[1] Although this knight's pattern is deterministic, by using the five different long streams of data generated by shuffling the trials, a more representative distribution of online learning performance was calculated.

For selective Kanerva coding, 5 different seeds were used to generate prototypes ranging from 1000 to 20000 where each prototype was a point in the 2, 8, or 12 dimensional space. Euclidean distance was used to measure how close each prototype was to the observed state. A constant ratio $\eta = 0.025$ of features (rounded down) were active at anytime. The prediction performance and the computation time per timestep was recorded. To reveal the relation between the number of features that could be calculated and the calculation time per timestep, the number of features that could be calculated within 100 ms was also recorded.

4 RESULTS

Figure 3 compares the variation in both prediction error and computation time for tile coding and SKC as their number of features increases, up to a maximum computation time per step cutt-off of 100ms (selected in this study as the upper limit for real-time prosthesis control, c.f., [4]). After removing poor performing outliers due to small representations from the tile coding data which were orders of magnitude worse than the trend seen in Fig. 3, the data indicates that both tile coding and SKC have improved performance as the number of features increases up until an optimum point at which the error increased or the representation took too long to compute. The maximum number of features calculated within 100 ms for the 12, 8, and 2 dimensional settings for SKC are represented as points I, II, and III, respectively. The mean number of features that can be calculated within 100 ms using tile coding had little variation on the log scale presented in Fig. 3, and is represented by a single point IV.

The 100 ms processing limit had the effect that 8 and 12 dimensional settings of tile coding did not improve beyond what the 2 dimensional setting was able to, despite the possibility of improved prediction performance if more computation was allowed. TC was quite unlike SKC, which not only had similar performance trends across different dimensions but utilized the additional dimensions of the state space to have statistically significant improvements in performance using the same number of prototypes and thus features. The best performance in terms of error across all conditions was found to be SKC at 8000 prototypes for all three different dimensionality conditions, with 12D SKC at 8000 features demonstrating the overall best performance on the entire experiment. Additionally, both tile coding and SKC had increased time step duration as the number of features increased.

---

[1] Similar to that in chess, the knight's pattern used here shifted the tilings over and up by an increasing amount for each new tiling, as per Miller and Glanz [19].

5 CONCLUSIONS

As the number of sensors available on prosthetic devices grows with their capabilities, an appropriate synthesis of sensorimotor signals into a useful representation becomes essential to the performance of these devices' machine learning control systems. If such a representation can remain computationally efficient, it can readily be used on the computationally limited systems residing within wearable prosthetic technology. The presented results reaffirm previous findings about tile coding's increasing computational requirements on high dimensional data and further show that selective Kanerva coding has better accuracy on an upper-limb robotic prediction task than tile coding. We note that selective Kanerva coding takes additional time to compute a representation, but also show that not only are there significant gains in prediction performance with additional features but that there is a performance gradient for a fixed activation ratio, η. These findings suggest that selective Kanerva coding merits further study, and as such, this work contributes a significant step towards accurately representing the high-dimensional data of assistive technologies to a machine learning control system such as a reinforcement learning agent.

References

[1] C. Castellini, P. Artemiadis, M. Wininger, et al., "Proceedings of the first workshop on peripheral machine interfaces: Going beyond traditional surface electromyography," *Frontiers in Neurorobotics*, vol. 8, no. 22, Aug. 2014. doi: 10.3389/fnbot.2014.00022.

[2] E. Scheme, K. Englehart, "Electromyogram pattern recognition for control of powered upper-limb prostheses: State of the art and challenges for clinical use," *J. of Rehabilitation Research and Development* vol. 48, no. 6, pp. 643–660, 2011.

[3] M. R. Dawson, C. Sherstan, J. P. Carey, et al., Development of the Bento Arm: An improved robotic arm for myoelectric training and research, *Proc. of MEC'14*, Fredericton, New Brunswick, August 18–22, 2014, pp. 60–64.

[4] T. R. Farrell, R. F. Weir, "The optimal controller delay for myoelectric prostheses," *IEEE Transactions on neural systems and rehabilitation engineering* vol. 15, no. 1, pp. 111–118, 2007.

[5] P. M. Pilarski, M. R. Dawson, T. Degris, et al., "Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning," *Proceedings of the 2011 IEEE International Conference on Rehabilitation Robotics (ICORR)*, June 29-July 1, Zurich, Switzerland, pp. 134–140. doi:10.1109/ICORR.2011.5975338

[6] P. M. Pilarski, M. R. Dawson, T. Degris, et al., "Adaptive artificial limbs: A real-time approach to prediction and anticipation," *IEEE Robotics and Automation Magazine* vol. 20, no. 1, pp. 53–64, 2013. doi: 10.1109/MRA.2012.2229948

[7] P. M. Pilarski, T. B. Dick, R. S. Sutton, "Real-time prediction learning for the simultaneous actuation of multiple prosthetic joints," *Proceedings of the 13th IEEE International Conference on Rehabilitation Robotics (ICORR)*, June 24-26, 2013, Seattle, USA, pp. 1–8. doi: 10.1109/ICORR.2013.6650435

[8] A. L. Edwards, J. S. Hebert, P. M. Pilarski, "Machine learning and unlearning to autonomously switch between the functions of a myoelectric arm," *Proceedings of the 6th IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob2016)*, June 26–29, 2016, Singapore, pp. 514-521.

[9] A. L. Edwards, M. R. Dawson, J. S. Hebert, et al., "Application of real-time machine learning to myoelectric prosthesis control: A case series in adaptive switching," *Prosthetics & Orthotics Int.*. vol. 40, no. 5, pp. 573–581, 2016.

[10] C. Sherstan, J. Modayil, P. M. Pilarski, "A collaborative approach to the simultaneous multi-joint control of a prosthetic Arm," *Proc. 14th IEEE/RAS-EMBS Int. Conf. on Rehabilitation Robotics (ICORR)*, August 11-14, 2015, Singapore, pp. 13-18.

[11] A. L. Edwards, "Adaptive and Autonomous Switching: Shared Control of Powered Prosthetic Arms Using Reinforcement Learning," M.Sc. Thesis, University of Alberta, 2016.

[12] R. S. Sutton, A. G. Barto. *Reinforcement learning: An introduction.* MIT press, Cambridge, 1998.

[13] R. S. Sutton, J. Modayil, M. Delp, et al., "Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction," *Proc. the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2–6, Taipei, Taiwan, 2011, pp. 761–768.

[14] H. van Seijen, A. R. Mahmood, P. M. Pilarski, et al., "True online temporal-difference learning," *Journal of Machine Learning Research* vol. 17, no. 14, pp. 1–40, 2016.

[15] W. Cheng, W. M. Meleis, "Adaptive Kanerva-based function approximation for multi-agent systems," *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1361-1364, 2008.

[16] B. Ratitch, D. Precup, "Sparse distributed memories for on-line value-based reinforcement learning," *European Conference on Machine Learning*. vol. 3201, Springer Berlin Heidelberg, 2004, pp. 347-358.

[17] B. Ratitch, et al., "Sparse distributed memories in reinforcement learning: Case studies.", *Proc. of the Workshop on Learning and Planning in Markov Processes-Advances and Challenges*, 2004, pp. 85-90.

[18] C. A. R. Hoare, "Algorithm 65: Find," *Communications of the ACM* vol. 4, iss. 7, pp. 321-322, 1961.

[19] W. T. Miller, F. H. Glanz, "UNH CMAC version 2.1: The University of New Hampshire implementation of the Cerebellar Model Arithmetic Computer - CMAC," Robotics Laboratory Technical Report, University of New Hampshire, Durham, New Hampshire, 1996.

[20] R. S. Sutton, et al., "Fast gradient-descent methods for temporal-difference learning with linear function approximation," *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 993-1000.

New Reinforcement Learning Using a Chaotic Neural Network for Emergence of "Thinking"
— "Exploration" Grows into "Thinking" through Learning —

Katsunari Shibata[*] **& Yuki Goto**
Department of Innovative (Electrical and Electronic) Engineering
Oita University
700 Dannoharu, Oita 870-1192, JAPAN
`katsunarishibata@gmail.com`

Abstract

Expectation for the emergence of higher functions is getting larger in the framework of end-to-end comprehensive reinforcement learning using a recurrent neural network. However, the emergence of "thinking" that is a typical higher function is difficult to realize because "thinking" needs non fixed-point, flow-type attractors with both convergence and transition dynamics. Furthermore, in order to introduce "inspiration" or "discovery" in "thinking", not completely random but unexpected transition should be also required.

By analogy to "chaotic itinerancy", we have hypothesized that **"exploration" grows into "thinking" through learning by forming flow-type attractors on chaotic random-like dynamics**. It is expected that if rational dynamics are learned in a chaotic neural network (ChNN), coexistence of rational state transition, inspiration-like state transition and also random-like exploration for unknown situation can be realized.

Based on the above idea, we have proposed **new reinforcement learning using a ChNN**. The positioning of exploration is completely different from the conventional one. Here, **the chaotic dynamics inside the ChNN produces exploration factors by itself**. Since external random numbers for stochastic action selection are not used, exploration factors cannot be isolated from the output. Therefore, the learning method is also completely different from the conventional one.

One variable named causality trace is put at each connection, and takes in and maintains the input through the connection according to the change in its output. Using these causality traces and TD error, the connection weights except for the feedback connections are updated in the actor ChNN.

In this paper, as the result of a recent simple task to see whether the new learning works appropriately or not, it is shown that a robot with two wheels and two visual sensors reaches a target while avoiding an obstacle after learning though there are still many rooms for improvement.

Keywords: thinking, chaotic neural network (ChNN), exploration, reinforcement learning (RL), function emergence

Acknowledgements

This research has been supported by JSPS KAKENHI Grant Numbers JP23500245, JP15K00360 and many our group members

---

[*] http://shws.cc.oita-u.ac.jp/˜shibata/home.html

1 Introduction

Expectation for the emergence of artificial intelligence is growing these days triggered by the recent results in reinforcement learning (RL) using a deep neural network (NN)[1, 2]. Our group has propounded for around 20 years that end-to-end RL from sensors to motors using a recurrent NN (RNN) plays an important role for the emergence[3, 4]. Especially, different from "recognition" whose inputs are given as sensor signals or "control" whose outputs are given as motor commands, higher functions are very difficult to design by human hands, and the function emergence approach through end-to-end RL is highly expected. Our group has shown that not only recognition and motion, but also memory, prediction, individuality, and also similar activities to those in monkey brain at tool use emerge[4]. We have also shown that a variety of communications emerge in the same framework[5]. However, the emergence of what can be called "thinking" that is one of the typical higher functions has not been shown yet. In this paper, the difficulty of emergence of "thinking" is discussed at first. Then our hypothesis that "exploration" grows into "thinking" through learning is introduced[6]. To realize the hypothesis, the use of a chaotic NN (ChNN) in RL and new deterministic RL for it are introduced[6]. Finally, it is shown that the new RL works in a simple task[7] though that cannot be called "thinking" yet and there are still many rooms for improvement. No other works with a similar direction to ours have not been found.

2 Difficulty in Emergence of "Thinking"

The definition of "Thinking" must be varied depending on the person. However, we can "think" even when we close our eyes and ears, and what we think does not change randomly, but logically or rationally. Therefore, we hope many ones can agree that in order to realize "thinking", rational multi-stage or flow-type state transition should be formed.

As a kind of dynamic functions, we have shown that a variety of memory-required functions emerge in an RNN in simple tasks[4]. It is not so difficult to form memories as fixed-point convergence dynamics if the initial feedback connection weights are set such that the transition matrix for the connection is the identity matrix or close to it when being linearly approximated. That can also solve the vanishing gradient problem in error back propagation. However, state transition needs not only convergence dynamics for association, but also transition dynamics from one state to another.

Then, we employed a multi-room task in which an agent moves and when it pushes a button, one of the doors opens and then the agent can move to another room. The sensor signals are the inputs of the agent's RNN, and the RNN was trained based on RL from a reward when it reached the goal. We expected that the internal state changed drastically between before and after the door open though the difference in the sensor signals was not so large. After learning, a large change in the internal state could be observed in some degree, but the learning was very difficult[8].

Furthermore, when we "think", "inspiration" or "discovery", which is a kind of unexpected but not completely random and rational transition, must be also essential. The convergence and transition dynamics seem contradict at a glance, and it seems very difficult to form both dynamics from scratch in a regular RNN.

3 Chaos and Hypothesis: Growth from "Exploration" to "Thinking" through Learning

Suppose we are standing at a forked road as shown in Fig. 1. Usually, we choose one from two options: going right and going left. We do not care many other possible actions such as going straight and dancing. It is not motor(actuator)-level exploration, but higher exploration supported by some prior or learned knowledge[9]. Furthermore, at a fork, we may wonder such that "this path looks more rough, but that way looks to go away from the destination". This can be considered as a kind of "exploration" and also as a kind of "thinking". The place where the wave in mind occurs is inside the process before making a decision, and learning should be reflected largely on the exploration. The author's group has thought that exploration should be generated inside of a recurrent NN (RNN) that generates motion commands[10]. "Exploration" and "thinking" are both generated as internal dynamics as shown in Fig. 2. "Exploration" is more random-

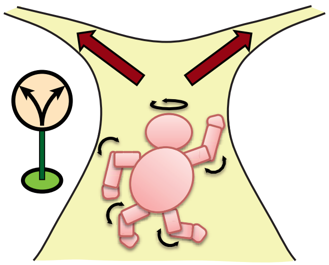


Figure 1: Lower or higher exploration at a fork.

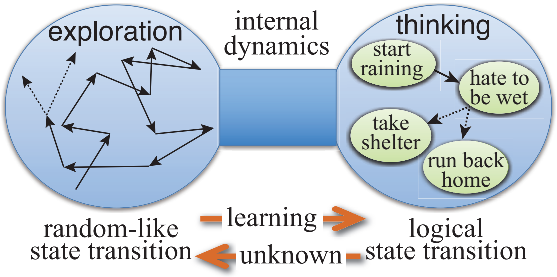


Figure 2: "Exploration" and "thinking" are both a kind of internal dynamics and deeply related with each other.

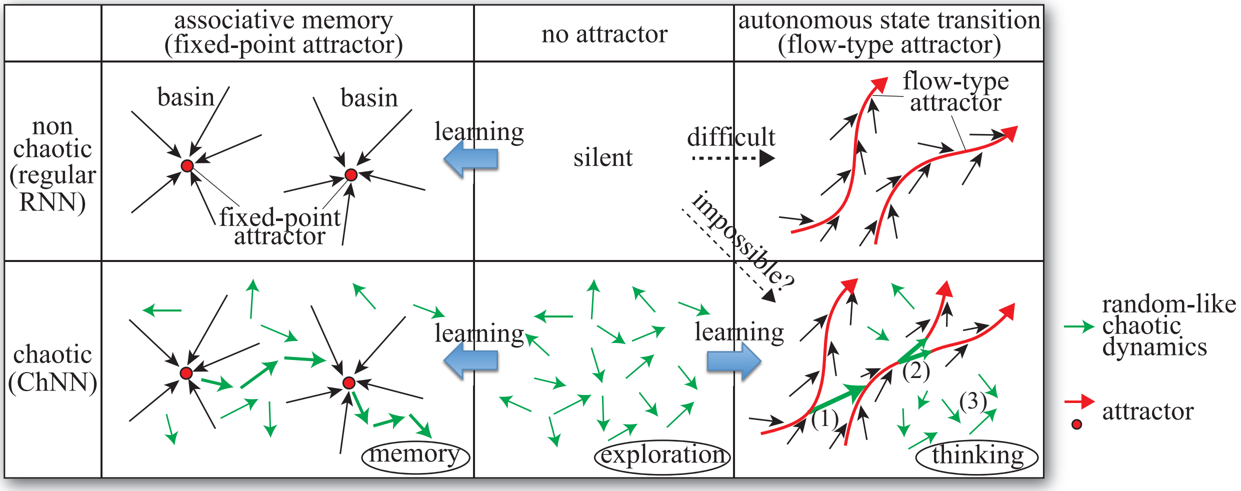


Figure 3: Rough schematic diagram of the combination of attractors and chaotic dynamics. See text in detail.

like state transitions. On the other hand, "thinking" is more rational or logical state transition, and sometimes higher exploration or unexpected but rational state transition such as inspiration or discovery occurs in it.

Analogy to such dynamics can be found in chaotic dynamics. In regular associative memory, fixed-point attractor (basin) is formed around each memorized or learned pattern as shown in the upper left part in Fig. 3. However, when a ChNN is used in it, transition dynamics among the memorized patters called "Chaotic Itinerancy"[11] can be seen as the green arrows in the lower left part in Fig. 3. If rational or logical state transitions are learned, it is expected that flow-type attractors are formed as the red or black arrows in the lower right part in Fig. 3. It is also expected that as the green arrows, (1)inspiration or discovery emerges as irregular transitions among the attractors but reflecting the distance, (2)higher exploration emerges at branches of the flow, and (3)in unknown situations, no attractor is formed, and remaining random-like chaotic dynamics appears until return to a known situation. Skarda et al. reported that the activities on the olfactory bulb in rabbits become chaotic for unknown stimuli[12]. Osana et al. have shown the difference of chaotic property between known and unknown patterns on an associative memory using a ChNN, and also after an unknown pattern is learned, association to the pattern is formed as well as the other known patterns[13]. From the above discussion, we have hypothesized that **"exploration" grows into "thinking" through learning by forming flow-type attractors on chaotic random-like dynamics and that can be realized on reinforcement learning using a ChNN**.

4 New Reinforcement Learning (RL) Using a Chaotic Neural Network (ChNN)

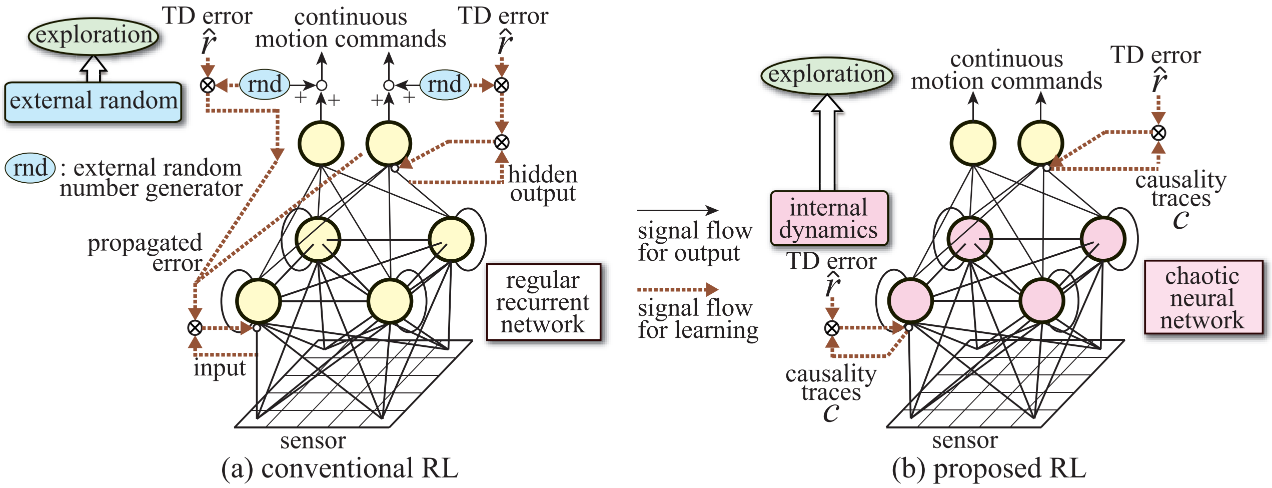
In order to realize the above idea, our group has proposed new reinforcement learning using a ChNN[6]. The positioning of exploration in learning is completely different from the conventional one. Here, the chaotic dynamics inside the ChNN produces exploration factors by itself. Since external random numbers for stochastic action selection are not used, exploration factors cannot be isolated from the output. Then the learning method has to be completely different from the conventional one.

Assuming that the motions are continuous, actor-critic type reinforcement learning architecture is employed. Here, to isolate the chaotic dynamics from the critic, the actor is implemented in a ChNN and the critic is implemented in another regular layered NN as shown in Fig. 5. The inputs are the sensor signals for the both networks. Here, only the learning of actor ChNN that is largely different from the conventional reinforcement learning is explained comparing with the conventional one using Fig. 4. In our conventional works, as shown in Fig. 4(a), by adding a random number (noise) $rnd_{j,t}$ to each actor output, the agent explores. The actor network is trained by BPTT (Back Propagation Through Time) using the product of the random number and TD error as the error for each output of the RNN.

In the proposed method, there is no external random numbers added to the actor outputs. The network is a kind of RNN, but by setting each feedback connection to a large random value, it can produce chaotic dynamics internally. Because the learning of recurrent connections does not work well, only the connections from inputs to hidden neurons and from hidden neurons to output neurons are trained. One variable C_{ji} named causality trace is put on each connection, and takes in and maintains the input through the connection according to the change in its output as

$$C_{ji,t}^{[l]} = (1 - |\Delta x_{j,t}^{[l]}|)C_{ji,t-1}^{[l]} + \Delta x_{j,t}^{[l]} x_{i,t}^{[l-1]} \tag{1}$$

where $x_{j,t}^{[l]}$: output of the j-th neuron in the l-th layer at time t, $\Delta x_t = x_t - x_{t-1}$.



(a) conventional RL (b) proposed RL

Figure 4: Comparison of conventional RL and proposed RL(only actor network) [6]

Using the causality trace C_{ji} and TD error $\hat{r}_t$, the weight $w^{[l]}_{ji}$ from the i-th neuron in $(l{-}1)$-th layer to the j-th neuron in the l-th layer is updated with a learning rate η as

$$\Delta w^{[l]}_{ji,t} = \eta \hat{r}_t C^{[l]}_{ji,t}. \tag{2}$$

5 Learning of Obstacle Avoidance

Since the learning is completely different from the conventional one, it is necessary to show whether the learning works appropriately in a variety of tasks or not. It was already applied to several tasks[6][9]. In this paper, the result of a recent task in which a robot has two wheels and two visual sensors is shown[7]. As shown in Fig. 5, there is a 20×20 size of field, and its center is the origin of the field. What the robot has to do is to reach a goal while avoiding an obstacle. The goal is put on the location (0, 8) with radius r=1.0, and a robot (r=0.5) and an obstacle (r=1.5) are put randomly at each episode. The orientation of the robot is also decided randomly. Each of the two omnidirectional visual sensors has 72 cells, and catches only goal or obstacle respectively. Total of 144 sensor signals are the inputs of both critic and actor networks, and the right and left wheels rotate according to the two actor outputs. When the robot comes in the goal area, a reward is given, and when it collides with the obstacle, a small penalty is given. One episode is defined as until arrival to the goal or 1000 steps from start. The both NNs have three layers including input layer. The number of hidden neurons is 10 for critic network and 100 for actor ChNN.

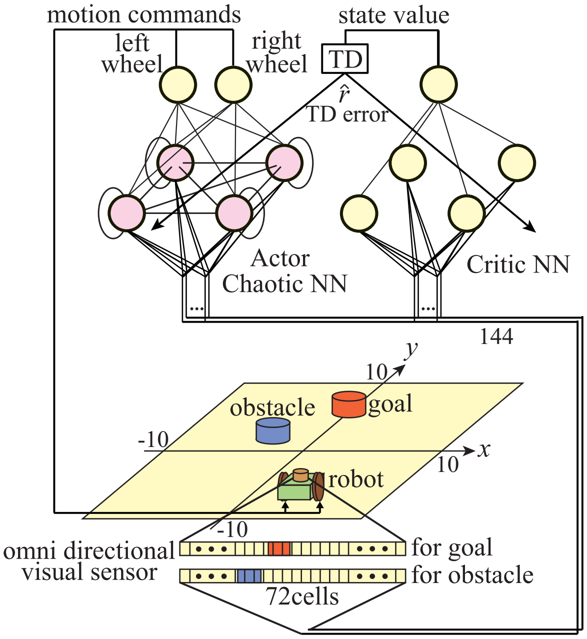
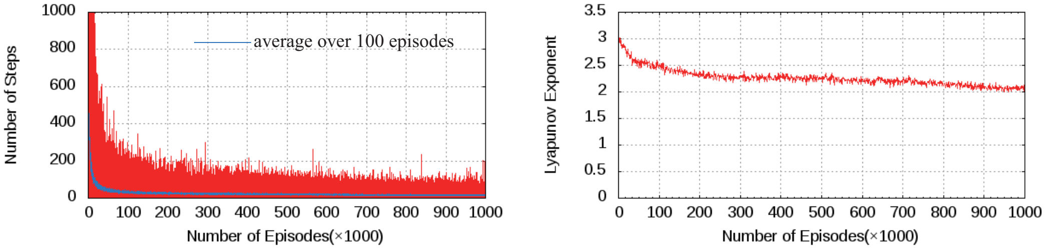


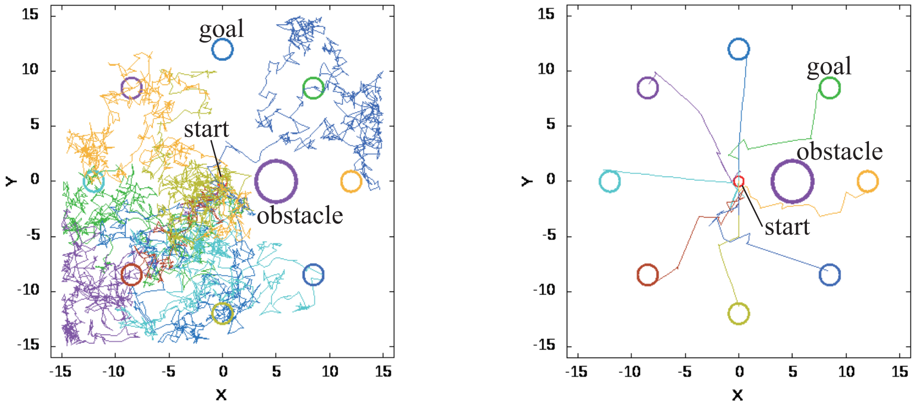
Figure 5: Learning of obstacle avoidance for a robot with two wheels and two sensors using a ChNN.

The learning results are shown in Fig. 6. Fig. 6(a) shows learning curve and the vertical axis indicates the number of steps to the goal. The blue line shows its average over each 100 episodes. It can be seen that according to the number of episodes, the number of steps decreases. Fig. 6(b) and (c) show the robot trajectories before and after learning respectively. Actually, the goal location is fixed and initial robot location is varied. But to see the initial robot orientation, the robot is on the center of the field, and its orientation is for upper-side initially on the figure. Relatively, the goal location is varied. The size of the obstacle in the figure shows the area where the robot collides with the obstacle and then is larger than the actual size. Before learning, the robot explored almost randomly, but after learning, it could reach the goal while avoiding the obstacle. In conventional learning, randomness of action selection is often decreased as learning progresses, but here, it can be seen that exploration factors decreased autonomously according to learning. However, there are some strange behaviors such



(a) Learning curve (Number of steps)

(d) Lyapunov exponent during learning



(b) Trajectories before learning

(c) Trajectories after learning

Figure 6: Learning results[7]. In (b) and (c), the initial robot location is on the center and its initial orientation is for upper-side in the figure. Actually, the goal is located at the same place, but on this figure the location is varied relatively.

that the robot changes its direction suddenly. It was observed that the actor hidden neurons were likely to have a value around the maximum 0.5 or minimum -0.5. There is still a large space to improve the learning method.

Fig. 6(d) shows Lyapunov exponent to see the chaotic property of this system including the environment. The robot is located at one of the 8 locations in Fig. 6(c), and the obstacle is also located at one of 8 locations. For each of the $8 \times 8 = 64$ combinations, a small perturbation vector with the size d_{before} is added to the internal states of hidden neurons, and the distance d_{after} in the internal states at the next time between the cases of no perturbation and addition of the perturbation is compared. The average of $ln(d_{after}/d_{before})$ is used as the exponent here. From the figure, it can be seen that Lyapunov exponent is gradually decreased, but since the value is more than 0.0, it is considered that the chaotic property is maintained. In [6], it was observed that when the environment changed, Lyapunov exponent increased again.

References

[1] Minh, V., Kavukcuoglu, K., Silver, D., et al. (2015) Human-level control through deep reinforcement learning, *Nature*, **518**, 529–533.

[2] Silver, D., Huang, A., et al. (2016) Mastering the game of Go with deep neural networks and tree search, *Nature*, **529**, 484–489.

[3] Shibata, K. (2011) Emergence of Intelligence through Reinforcement Learning ..., *Advances in Reinforcement Learning*, Intech, 99–120.

[4] Shibata, K. (2017) Functions that Emerge through End-to-end Reinforcement Learning, *RLDM 2017*.

[5] Shibata, K. (2017) A Variety of Communications that Emerge through Reinforcement Learning Using ..., *RLDM 2017*.

[6] Shibata, K., & Sakashita, Y. ('2015) Reinforcement Learning with Internal-Dynamics-based Exploration Using a Chaotic Neural Network, *Proc. of IJCNN (Int'l Joint Conf. on Neural Networks) 2015*, #15231.

[7] Shibata, K. & Goto, Y. (2017) Significance of Function Emergence Approach based on End-to-end Reinforcement Learning as suggested by Deep Learning, and Novel Reinforcement Learning Using a Chaotic ..., *Cognitive Studies*, **24** (1), 96–117. (in Japanese)

[8] Sawatsubashi, Y., et al. (2012) Emergence of Discrete and Abstract State ..., *Robot Intelligence Tech. and App. 2012*, 13–22.

[9] Goto, Y., & Shibata, K. (2016) Emergence of Higher Exploration in Reinforcement Learning Using a Chaotic Neural Network, *Neural Information Processing, Lecture Notes in Computer Science*, **9947**, 40–48.

[10] Shibata, K. (2006) Learning of Deterministic Exploration and Temporal Abstraction in ..., *Proc. of SICE-ICCAS*, 4569–4574.

[11] Kaneko, K. & Tsuda, I.(2003) Chaotic itinerancy, *Chaos*, **13**(3), 926-936.

[12] Skarda, C. A., & Freeman, W. J. (1987) How brains make chaos in order to make sense ..., *Behavioral and Brain Science*, **10**, 161–173.

[13] Osana, Y. & Hagiwara, M. (1999) Successive Learning in hetero-associative memory using chaotic neural networks. *Int. J. Neural Systems*, **9**(4):285- 299.

Enhancing metacognitive reinforcement learning using reward structures and feedback

Paul M. Krueger [1]
Dept. of Psychology, UC Berkeley
pmk@berkeley.edu

Falk Lieder [1]
Dept. of Psychology, UC Berkeley
falk.lieder@berkeley.edu

Thomas L. Griffiths
Dept. of Psychology, UC Berkeley
tom_griffiths@berkeley.edu

[1] **These authors contributed equally.**

Abstract

One of the most remarkable aspects of the human mind is its ability to improve itself based on experience. Such learning occurs in a range of domains, from simple stimulus-response mappings, motor skills, and perceptual abilities, to problem solving, cognitive control, and learning itself. Demonstrations of cognitive and brain plasticity have inspired cognitive training programs. The success of cognitive training has been mixed and the underlying learning mechanisms are not well understood. Feedback is an important component of many effective cognitive training programs, but it remains unclear what makes some feedback structures more effective than others. To address these problems, we model cognitive plasticity as metacognitive reinforcement learning. Here, we develop a metacognitive reinforcement learning model of how people learn how many steps to plan ahead in sequential decision problems, and test its predictions experimentally. The results of our first experiment suggested that our model can discern which reward structures are more conducive to metacognitive learning. This suggests that our model could be used to design feedback structures that make existing environments more conducive to cognitive growth. A follow-up experiment confirmed that feedback structures designed according to our model can indeed accelerate learning to plan. These results suggest that modeling metacognitive learning is a promising step towards building a theoretical foundation for promoting cognitive growth through cognitive training and other interventions.

Keywords: Decision-Making; Planning; Metacognitive Reinforcement Learning; Cognitive Training

Acknowledgements

This work was supported by grant number ONR MURI N00014-13-1-0341. The authors thank Thomas Hills, Peter Dayan, Rika Antonova, Silvia Bunge, Stuart Russell, Smitha Milli, Amitai Shenhav, and Sebastian Musslick for feedback and discussions.

1 Introduction and background

How do we learn to think better, and what can we do to promote such metacognitive learning? Demonstrations of cognitive and brain plasticity have inspired cognitive training programs. Yet, the success of cognitive training has been mixed and the underlying learning mechanisms are not well understood (Owen et al., 2010; Morrison & Chein, 2011). Feedback is an important component of many effective cognitive training programs, but it remains unclear what makes some feedback structures more effective than others, and there is no principled method for designing optimal feedback structures.

Here, we propose that cognitive growth proceeds through metacognitive reinforcement learning. We apply this theory to model how people learn how far to plan ahead, test its predictions experimentally, and apply it to design feedback structures that accelerate metacognitive learning.

Planning problems are commonly formulated as a *Markov decision process* (MDP), $M = (\mathcal{S}, \mathcal{A}, T, \gamma, r, P_0)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $T(s, a, s')$ is the probability that the agent will transition from state s to state s' if it takes action a, $0 \leq \gamma \leq 1$ is the discount factor, $r(s, a, s')$ is the reward generated by this transition, and P_0 is the probability distribution of the initial state S_0 (Sutton & Barto, 1998). A *policy* $\pi : \mathcal{S} \mapsto \mathcal{A}$ specifies which action to take in each of the states. The expected sum of discounted rewards that a policy π will generate in the MDP M starting from a state s is known as its *value function* $V_M^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r\left(S_t, \pi(S_t), S_{t+1}\right)\right]$. The optimal policy $\pi_M^\star$ maximizes the expected sum of discounted rewards, that is $\pi_M^\star = \arg\max_\pi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r\left(S_t, \pi(S_t), S_{t+1}\right)\right]$.

Solving large planning problems is often intractable. However, planners can save a substantial amount of time and effort by finding the plan that maximizes their expected reward over the next one, two, or three steps, instead of over a lifetime. Since cutting corners in the decision process is both necessary and problematic, good decision-making requires knowing when that is admissible and when it is not. Previous work suggests that this metacognitive skill can be learned through trial and error (Lieder & Griffiths, 2015), which can be understood in terms of *reinforcement learning* (Sutton & Barto, 1998). While certain reinforcement learning algorithms can, in principle, learn to solve arbitrarily complex problems, reinforcement learning can also be very slow—especially when rewards are sparse and the optimal policy is far from the learner's initial strategy. A common approach to remedy this problem is to give the algorithm pseudo-rewards for actions that do not achieve the goal but lead in the right direction (Ng, Harada, & Russell, 1999).

2 Deciding how to decide

People can use many different decision strategies. This poses the problem of deciding how to decide (Boureau, Sokol-Hessner, & Daw, 2015). Previous research on meta-decision-making has focused on the arbitration between habits versus planning (Keramati, Dezfouli, & Piray, 2011; Dolan & Dayan, 2013). While this is an important meta-control problem, it is only one part of the puzzle because people are equipped with more than one goal-directed decision-mechanism. Hence, when the model-based system is in charge, it has to be determined how many steps it should plan ahead. Ideally, the chosen planning horizon should achieve the optimal tradeoff between expected decision quality versus decision time and mental effort (Shenhav et al., 2017).

Here, we make the simplifying assumption that people always choose the action that maximizes their sum of expected rewards over the next h steps, for some value of h that differs across decisions. Under this assumption, the meta-decision problem is to select a planning horizon h from a set $\mathcal{H} = \{1, 2, \cdots, \}$, execute the plan, select a new planning horizon, and so on. More formally, this problem can be formalized as a meta-level MDP (Hay, Russell, Tolpin, & Shimony, 2012). In our task, the meta-level MDP is $M_{\text{meta}} = (\mathcal{S}_{\text{meta}}, \mathcal{H}, T_{\text{meta}}, r_{\text{meta}})$, where the meta-level state $m \in \mathcal{S}_{\text{meta}} = \{0, 1, 2, 3, 4\}$ encodes the number of remaining moves, and the meta-level action $h \in \mathcal{H} = \{1, 2, 3, 4\}$ is the planning horizon used to make a decision. The meta-level reward function r_{meta} integrates the cost of planning with the return of the resulting action: $r_{\text{meta}}(m_k, h_k) = -\text{cost}(h_k) + \sum_{t=1}^{h} r(s_t, \text{plan}_t^{(k,h_k)})$, where $\text{plan}_t^{(k,h)}$ is the tth action of the plan formed by looking h steps ahead in the meta-level state m_k. The meta-decision-maker receives this reward after the plan has been executed in its entirety. This happens multiple times within each problem unless the meta-decision maker chooses to plan l or more steps ahead at the beginning of an l-step problem. The cost of planning $\text{cost}(h_k)$ is determined by the branching factor b, the length of the plan h, and the planning cost per planning step (λ), that is $\text{cost}(h_k) = \lambda \cdot b^{h_k} \cdot h_k$.

3 Metacognitive reinforcement learning

Solving the problem of deciding how to decide optimally is computationally intractable but the optimal solution can be approximated through learning (Russell & Wefald, 1991). We propose that people use reinforcement learning (Sutton & Barto, 1998) to approximate the optimal solution to the meta-decision problem formulated above.

Our model of metacognitive reinforcement learning builds on the semi-gradient SARSA algorithm (Sutton & Barto, 1998) that was develop to approximately solve MDPs with large or continuous state spaces. Specifically, we assume that people learn a linear approximation to the meta-level Q-function $Q_{\text{meta}}(m_k, h_k) \approx \sum_{j=1}^{7} w_j \cdot f_j(m_k, h_k)$, whose features $\mathbf{f}$ comprise one indicator variable for each possible planning horizon h ($f_1 = \mathbb{1}(h = 1), \cdots, f_4 = \mathbb{1}(h = 4)$), one indicator variable for whether or not the agent planned all l steps until the end of the task ($f_5 = \mathbb{1}(h = l)$), the number of steps that were left unplanned ($f_6 = \max\{0, l - h\}$), and the number of steps the agent planned too far ($f_7 = \max\{0, h - l\}$). The semi-gradient SARSA algorithm learns the weights of these features by gradient descent. To bring it closer to human performance, our model replaces its gradient descent updates by Bayesian learning. Concretely, the weights $\mathbf{w}$ are learned by Bayesian linear regression of the bootstrap estimate $\hat{Q}(m_k, h_k)$ of the meta-level value function onto the features $\mathbf{f}$. The bootstrap estimator $\hat{Q}(m_k, h_k) = r_{\text{meta}}(m_k, h_k) + \langle \mu_t, \mathbf{f}(m', h') \rangle$ is the sum of the immediate meta-level reward and the predicted value of the next meta-level state m'. The predicted value of m' is the scalar product of the posterior mean μ_t of the weights $\mathbf{w}$ given the observations from the first t actions (where $t = \sum_{n=1}^{k} h_n$) and the features $\mathbf{f}(m', c')$ of m' and the planning horizon h' that will be selected in that state.

We assume that the prior on the feature weights reflects that it is beneficial to plan until the end ($P(f_5) = \mathcal{N}(\mu = 1, \sigma = 0.1)$), although planning is costly ($P(f_1) = P(f_2) = P(f_3) = P(f_4) = \mathcal{N}(\mu = -1, \sigma = 0.1)$), and that planning too much is more costly than planning too little ($P(f_7) = \mathcal{N}(\mu = -1, \sigma = 0.1)$ and $P(f_6) = \mathcal{N}(\mu = 0, \sigma = 0.1)$).

Given the learned posterior distribution on the feature weights $\mathbf{w}$, the planning horizon h is selected by Thompson sampling. Specifically, to make the kth meta-decision, a weight vector $\tilde{w}$ is sampled from the posterior distribution of the weights given the series of meta-level states, selected planning horizons, and resulting value estimates experienced so far, that is $\tilde{w}_k \sim P(\mathbf{w}|\mathcal{E}_k)$, where the set $\mathcal{E}_k = \{e_1, \cdots, e_k\}$ contains the meta-decision-maker's experience from the first k meta-decisions; to be precise, each meta-level experience $e_j \in \mathcal{E}_k$ is a tuple $\left(m_j, h_j, \hat{Q}(m_j, c_j; \mu_j)\right)$ containing a meta-level state, the computation selected in it, and the bootstrap estimates of its Q-value. The sampled weight vector $\tilde{w}$ is then used to predict the Q-values of each possible planning horizon $h \in \mathcal{H}$. Finally, the planning horizon with the highest predicted Q-value is used for decision-making.

By proposing metacognitive reinforcement learning as a mechanism of cognitive plasticity, our model suggests that reward and feedback are critical for cognitive growth. Conceptualizing metacognitive reinforcement learning as a regression problem suggests that some reward structures should be more conducive to cognitive growth than others. Here, we apply our model to predict how quickly people can learn that more planning leads to better results from the reward structure of the practice problems. According to the model, learning should be fastest when the reward increases deterministically with the planning horizon both within and across problems. By contrast, learning should be slower when this relationship is degraded by additional variability in the rewards that is unrelated to planning. The following experiments test this prediction and illustrate the model's utility for designing feedback structures that promote metacognitive learning.

4 Experiment 1: Reward structures can help or hinder learning to plan

4.1 Methods

We recruited 304 adult participants from Amazon Mechanical Turk. The task took about 25 minutes, and participants were paid \$2.50 plus a performance-dependent bonus of up to \$2.00. Participants played a series of *flight planning* games. The environment consisted of six different cities, each connected to two other cities. Participants began each trial at a given city, and were tasked with planning a specified number of flights. Each flight was associated with a known gain or loss of money, displayed onscreen. Thus, the participants' task was to plan a route that would maximize their earnings or minimize their losses, based on the number of planning steps required for that game.

The experiment comprised thirteen trials in total: a sequence of three practice problems which required planning 2, 3, and 3 steps ahead, respectively, followed by ten 4-step problems. Participants were assigned randomly to one of two conditions: environments with reward structures designed to promote learning ("diagnostic rewards"), or environments with reward structures designed to hinder learning ("non-diagnostic rewards").

The problems of the diagnostic rewards condition were automatically generated to exhibit four characteristics:

1. If the number of moves is l then planning $h < l$ steps ahead generates $l - h$ suboptimal moves. In other words, each myopic planner makes the maximum possible number of mistakes.
2. When the number of moves is l, then planning l steps ahead yields a positive return, but planning $h < l$ steps ahead yields a negative return.
3. The return increases monotonically with the planning horizon from 1 to the total number of moves.
4. Each starting position occurs at least once.

The reward structures used for the non-diagnostic rewards condition were created by shifting the diagnostic reward structures so as to degrade the correlation between planning horizon and reward.

4.2 Results

Both model simulations and human behavior demonstrated enhanced learning in environments with diagnostic rewards. Figure 1 shows the mean performance of the metacognitive reinforcement learning model, and the mean performance of human participants. Here, performance is measured as relative reward $R_{rel} = (R - R_{min})/(R_{max} - R_{min})$, where R is the total reward received during the trial, and R_{min} and R_{max} are the highest and lowest possible total reward on that trial, respectively.

To measure the effects of condition and trial number on performance in human participants, we ran a repeated-measures ANOVA. This revealed a significant effect of both trial number ($F(9, 2989) = 3.44, p < 0.001$) and condition ($F(9, 3029) = 15.26, p < 0.0001$), such that participants improved over time, and participants with diagnostic feedback performed better than those without. To measure learning in each group, we ran a simple linear regression of the relative reward on the trial number. This revealed a significant correlation for participants who received diagnostic rewards ($F(2, 302) = 11.28, p < 0.01$), with an R^2 of 0.59, but not for participants who received non-diagnostic rewards ($F(2, 302) = 3.51, p > 0.05$), with an R^2 of 0.31, suggesting that improvement in performance occurred with diagnostic rewards, but not without.

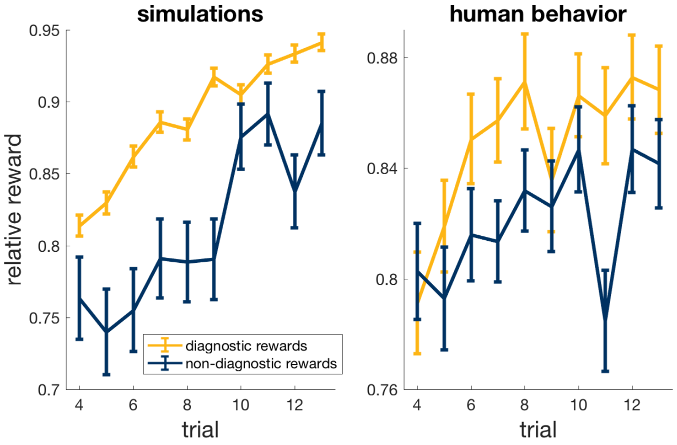


Figure 1: Model predictions (averaged over 500 simulation) and human performance in Experiment 1 ±1SEM.

We performed a multinomial logistic regression of whether or not each participant chose the optimal route on trial number and group. This revealed significant effects of trial number ($p < 10^{-6}$) and group ($p < 0.0001$).

In addition, we found that participants interacting with a diagnostic reward structure learned to plan significantly further ahead than participants interacting with the non-diagnostic reward structure. When there were four steps left, the average planning horizon was 2.96 with diagnostic rewards compared to 2.65 with non-diagnostic rewards ($t(596) = 2.94, p < 0.01$). When the rewards were diagnostic of good planning, participants' choices in the first step of the 4-step problems accorded 10.3% more frequently with 4-step planning ($t(302) = 3.57, p < 0.001$). For 3 remaining steps there was a significant increase in choices according with optimal 1-step ($p < 0.01$), 2-step ($p < 0.01$) and 4-step planning ($p < 0.01$). For 2 remaining steps, there was a significant increase in choices according with optimal 1-step planning ($p < 0.0001$) without a decrease in agreement with other planning horizons. Finally, on the last move participants' choices in the environment with diagnostic rewards corresponded 5.8% more frequently with optimal 1-step planning ($t(302) = 3.71, p < 0.001$), and significantly less frequently with 2-step and 3-step planning ($p < 0.01$ and $p < 0.001$). In summary, diagnostic rewards led to better agreement between the planning horizon and the number of remaining steps.

5 Experiment 2: Using feedback to promote learning to plan

In Experiment 2, we tested whether providing feedback may be an effective alternative approach to accelerating learning.

5.1 Methods

324 adult participants played twenty trials of the flight planning game described above. Participants in the feedback condition were told their apparent planning horizon at the end of every trial and penalized with a timeout that reflected the amount of planning they had eschewed. Concretely, we set the durations of the timeouts such that the cost of short-sighted decisions was proportional to the amount of necessary planning the participant had eschewed: cost $= 2^{l-\hat{h}}$, where l is the number of moves for that trial, $\hat{h}$ is the participant's apparent planning horizon, and 2 is the branching factor. The participant's planning horizon was estimated by the number of consecutive moves consistent with the optimal policy, beginning with the last move. At the end of each trial participants in the feedback group were penalized with a timeout delay for sub-optimal routes, calculated as $7 \cdot (\text{cost} - 1)$ seconds. The control group received no feedback on their

apparent planning horizon and had to wait 8 seconds at the end of every trial, to control the overall timeout duration across groups.

5.2 Model Predictions

We applied the metacognitive reinforcement learning model described above to the problem of learning how many steps one should plan ahead. We simulated a run of the experiment described above with 1000 participants in each condition. The simulations predicted a gradual increase in the relative return from the first 3-step problem to the last one. With feedback, the relative return increased faster and reached a higher level than without feedback.

5.3 Results

The pattern of qualitative and quantitative results in Experiment 2 followed a similar pattern as in Experiment 1: a significant effect of feedback ($F(9, 4521) = 8.54, p < 0.01$) and trial number ($F(9, 4521) = 1.85, p < 0.05$) on relative reward; and a significant regression equation for the feedback group ($F(2, 322) = 5.28, p = 0.05$, $R^2 = 0.40$) but not the control group ($F(2, 322) = 1.57, p > 0.05$, $R^2 = 0.16$), suggesting learning with, but not without, feedback.

As predicted by our model, a multinomial logistic regression of whether or not each participant chose the optimal route on trial number and feedback, revealed significant effects of trial number ($p < 0.0001$) and feedback ($p < 0.01$).

Feedback appeared to increase people's planning horizons: when there were two remaining moves, the choices of the feedback group accorded 4% less often with myopic choice ($t(1398) = -2.17, p < 0.05$), 7% more often with optimal 2-step planning ($t(1398) = 3.44, p < 0.001$), and 4% more often with optimal 3-step planning ($t(1398) = 2.43, p < 0.05$).

6 Discussion

In this article, we have introduced a computational model of how people learn to decide better. Its central idea is that learning how to think can be understood as metacognitive reinforcement learning. Our model extends previous research on strategy selection learning (Lieder et al., 2014; Lieder & Griffiths, 2015) by capturing that choosing cognitive operations is a sequential decision problem with potentially delayed rewards rather than a one-shot decision. The new model correctly predicted the effects of reward structure and feedback on learning to plan: Experiment 1 suggested that our model captures the effect of reward structures on the speed of metacognitive learning. We then applied our theory to design feedback for people's performance in environments whose reward structure is not diagnostic of good planning. Experiment 2 confirmed the model's prediction that this intervention would be effective. We hope that our theory of metacognitive reinforcement learning will be a step towards establishing a scientific foundation for designing feedback for cognitive training and other interventions for promoting cognitive growth.

References

Boureau, Y.-L., Sokol-Hessner, P., & Daw, N. D. (2015). Deciding how to decide: self-control and meta-decision making. *Trends in cognitive sciences*, *19*(11), 700–710.

Dolan, R. J., & Dayan, P. (2013). Goals and habits in the brain. *Neuron*, *80*(2), 312–325.

Hay, N., Russell, S., Tolpin, D., & Shimony, S. (2012). Selecting computations: Theory and applications. In N. de Freitas & K. Murphy (Eds.), *Uncertainty in artificial intelligence: Proceedings of the twenty-eighth conference.* P.O. Box 866 Corvallis, Oregon 97339 USA: AUAI Press.

Keramati, M., Dezfouli, A., & Piray, P. (2011). Speed/accuracy trade-off between the habitual and the goal-directed processes. *PLoS Comput Biol*, *7*(5), e1002055.

Lieder, F., & Griffiths, T. L. (2015). When to use which heuristic: A rational solution to the strategy selection problem. In *Proceedings of the 37th annual conference of the cognitive science society.*

Lieder, F., Plunkett, D., Hamrick, J. B., Russell, S. J., Hay, N., & Griffiths, T. (2014). Algorithm selection by rational metareasoning as a model of human strategy selection. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, & K. Weinberger (Eds.), *Advances in neural information processing systems 27* (pp. 2870–2878). Curran Associates, Inc.

Morrison, A. B., & Chein, J. M. (2011). Does working memory training work? the promise and challenges of enhancing cognition by training working memory. *Psychonomic bulletin & review*, *18*(1), 46–60.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In I. Bratko & S. Dzeroski (Eds.), *Proceedings of the 16th Annual International Conference on Machine Learning* (pp. 278–287). San Francisco: Morgan Kaufmann.

Owen, A. M., Hampshire, A., Grahn, J. A., Stenton, R., Dajani, S., Burns, A. S., … Ballard, C. G. (2010). Putting brain training to the test. *Nature*, *465*(7299), 775–778.

Russell, S., & Wefald, E. (1991). Principles of metareasoning. *Artificial Intelligence*, *49*(1-3), 361–395.

Shenhav, A., Musslick, S., Lieder, F., Kool, W., Griffiths, T., Cohen, J., & Botvinick, M. (2017). Toward a rational and mechanistic account of mental effort. *Annual Review of Neuroscience*, *40*.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA, USA: MIT press.

Speeding Up HAM Learning with Internal Transitions

Aijun Bai
University of California, Berkeley
Berkeley, CA 94720
`aijunbai@berkeley.edu`

Stuart Russell
University of California, Berkeley
Berkeley, CA 94720
`russell@cs.berkeley.edu`

Abstract

In the context of hierarchical reinforcement learning, the idea of *hierarchies of abstract machines* (HAMs) is to write a partial policy as a set of hierarchical finite state machines with unspecified choice states, and use reinforcement learning to learn an optimal completion of this partial policy. Given a HAM with potentially deep hierarchical structure, there often exist many internal transitions where a machine calls another machine with the environment state unchanged. In this paper, we propose a new hierarchical reinforcement learning algorithm that automatically discovers such internal transitions, and shortcircuits them recursively in the computation of Q values. We empirically confirm that the resulting HAMQ-INT algorithm outperforms the state of the art significantly on the benchmark Taxi domain and a much more complex RoboCup Keepaway domain.

Keywords: Hierarchical Reinforcement Learning, HAM

1 Introduction

Hierarchical reinforcement learning (HRL) aims to scale reinforcement learning (RL) by incorporating prior knowledge about the structure of good policies into the algorithms [Barto and Mahadevan2003]. Popular HRL solutions include the *options* theory [Sutton *et al.*1999], the *hierarchies of abstract machines* (HAMs) framework [Parr and Russell1998], and the *MAXQ* approach [Dietterich1999]. This paper describes a new HRL algorithm taking advantage of internal transitions introduced by the input hierarchical structure, following the framework of HAMs. It is our observation that a HAM with deep hierarchical structure, where there are many calls from a parent machine to one of its child machines over the hierarchy, induces many internal transitions. An internal transition is a transition over the joint state space, where only the run-time stack changes, but not the environment state. Internal transitions always come with zero rewards and deterministic outcomes in the resulting SMDP. It is usually the case that there could be many such internal transitions in a HAM with arbitrary structure for a complex domain. In the setting of concurrent learning when multiple HAMs are running concurrently, as shown in [Marthi *et al.*2005], there are even more opportunities of having internal transitions over the resulting joint SMDP.

The resulting HAMQ-INT algorithm identifies and takes advantage of internal transitions within a HAM for efficient learning. HAMQ-INT recursively shortcircuits the computation of Q values whenever applicable. We empirically confirm that HAMQ-INT outperforms the state of the art significantly on the benchmark Taxi domain and a much more complex RoboCup Keepaway domain. The two contributions of this paper is that 1) we develop the novel HAMQ-INT algorithm, and 2) we apply it successfully to the RoboCup Keepaway domain, which, to the best of our knowledge, is the first application of the HAM framework to a very complex domain.

2 The HAM Framework

The idea of HAM is to encode a partial policy for an agent as a set of hierarchical finite state machines with unspecified choice states, and use RL to learn its optimal completion. We adopt a different definition of HAM, allowing arbitrary call graph, despite the original notation of [Parr and Russell1998] which requires that the call graph is a tree. Formally, a HAM $\mathcal{H} = \{\mathcal{N}_0, \mathcal{N}_1, \dots\}$ consists of a set of Moore machines $\mathcal{N}_i$, where $\mathcal{N}_0$ is the root machine which serves as the starting point of the agent. A machine $\mathcal{N}$ is tuple $\langle M, \Sigma, \Lambda, \delta, \mu \rangle$, where M is the set of machine states, Σ is the input alphabet which corresponds to the environment state space S, Λ is the output alphabet, δ is the machine transition function with $\delta(m, s)$ being the next machine state given machine state $m \in M$ and environment state $s \in S$, and μ is the machine output function with $\mu(m) \in \Lambda$ being the output of machine state $m \in M$. There are 5 types of machine states: **start** states are the entries of running machines; **action** states execute an action in the environment; **choose** states nondeterministically select the next machine states; **call** states invoke the execution of other machines; and, **stop** states end current machines and return control to calling machines. A machine $\mathcal{N}$ has uniquely one **start** state and one **stop** state, referred as $\mathcal{N}.start$ and $\mathcal{N}.stop$ respectively. For **start** and **stop** states, the outputs are not defined; for **action** states, the outputs are the associated primitive actions; for **call** states, the outputs are the next machines to run; and, for **choose** states, the outputs are the sets of possible choices, where each choice corresponds to a next machine state.

To run a HAM $\mathcal{H}$, a run-time stack (or stack for short) is needed. Each frame of this stack stores run-time information such as the active machine, its machine state, the parameters passing to this machine and the values of local variables used by this machine. Let $\mathcal{Z}$ be the space of all possible stacks given HAM $\mathcal{H}$. It has been shown that an agent running a HAM $\mathcal{H}$ over an MDP $\mathcal{M}$ yields a joint SMDP $\mathcal{H} \circ \mathcal{M}$ defined over the joint space of S and $\mathcal{Z}$. The only actions of $\mathcal{H} \circ \mathcal{M}$ are the choices allowed at choice points. A choice point is a joint state (s, z) with $z.$`Top()` being a **choose** state. This is an SMDP because once a choice is made at a choice point, the system — the composition of $\mathcal{H}$ and $\mathcal{M}$ — runs automatically until the next choice point is reached. An optimal policy of this SMDP corresponds to an optimal completion of the input HAM, which can be found by applying a HAMQ algorithm [Parr and Russell1998]. HAMQ keeps track of the previous choice point (s, z), the choice made c and the cumulative reward r thereafter. Whenever it enters into a new choice point (s', z'), it performs the SMDP Q update as follows: $Q(s, z, c) \leftarrow (1 - \alpha)Q(s, z, c) + \alpha\,(r + \gamma^{\tau} \max_{c'} Q(s', z', c'))$, where τ is the number of steps between the two choice points. As suggested by the language of ALisp [Andre and Russell2002], a HAM can be equivalently converted into a piece of code in modern programming languages, with *call-and-return* semantics and built-in routines for explicitly updating stacks, executing actions and getting new environment states. The execution of a HAM can then be simulated by running the code itself.

3 The Main Approach

3.1 Internal Transitions within HAMs

In general, the transition function of the resulting SMDP induced by running a HAM has the form $T(s', z', \tau | s, z, c) \in [0, 1]$, where (s, z) is the current choice point, c is the choice made, (s', z') is the next choice point, and τ is the number of time steps. Given a HAM with potentially a deep hierarchy of machines, it is usually the case that there is no real

```
QUpdate($s'$ : state, $z'$ : stack, $r$ : reward, $t'$ : current time,
  $\mathcal{P}$ : evaluated predicates):
if $t' = t$ then
    $\rho[\mathcal{P}, \mathcal{P}(s), z, c] \leftarrow z'$
else
    QTable($s, z, c$) $\leftarrow (1 - \alpha)$ QTable($s, z, c$)
      $+\alpha(r + \gamma^{t'-t} \max_{c'}$ Q($s', z', c'$) $)$
$(t, s, z) \leftarrow (t', s', z')$
```

```
Q($s$ : state, $z$ : stack, $c$ : choice):
if $\exists \mathcal{P}$ s.t. $\langle \mathcal{P}, \mathcal{P}(s), z, c \rangle \in \rho$.Keys() then
    $q \leftarrow -\infty$
    $z' \leftarrow \rho[\mathcal{P}, \mathcal{P}(s), z, c]$
    for $c' \in \mu(z$.Top()$)$ do
        $q \leftarrow \max(q,$ Q($s, z', c'$) $)$
    return $q$
else
    return QTable($s, z, c$)
```

Figure 1: The HAMQ-INT algorithm.

actions executed between two consecutive choice points, therefore the number of time steps and the cumulative reward in-between are essentially zero. We call this kind of transitions internal transitions, because the machine state changes, but not the environment state. Formally, a transition is a tuple $\langle s, z, c, r, s', z' \rangle$ with r being the cumulative reward. For an internal transition, we must have $s' = s$ and $r = 0$. In addition, because the dynamics of the HAM after a choice has been made and before an action is executed is deterministic by design, the next choice point (s, z') of an internal transition is deterministically conditioned only on $\langle s, z, c \rangle$. Let $\rho(s, z, c)$ be the $\mathcal{Z}$ component of the next choice point. If $\langle s, z, c \rangle$ leads to an internal transition, we must have $T(s, \rho(s, z, c), 0 | s, z, c) = 1$. Therefore, we have

$$Q(s, z, c) = V(s, \rho(s, z, c)) = \max_{c'} Q(s, \rho(s, z, c), c'). \tag{1}$$

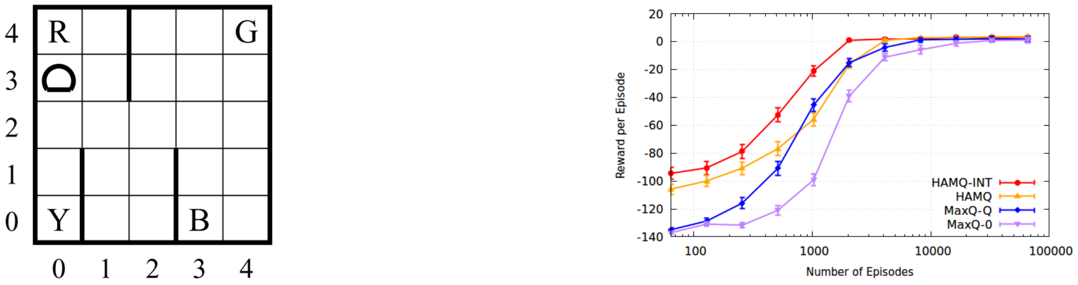
So, we can store the rules of internal transition as $\langle s, z, c, z' \rangle$ tuples, where $z' = \rho(s, z, c)$. They can be used to recursively compute Q values according to Equation 1 when applicable. The size of the set of stored rules can be further reduced, because the machine transition function δ of a HAM is usually determined by a set of predicates defined over environment state s, rather than the exact values of all state variables. Suppose $\langle s_1, z, c \rangle$ leads to an internal transition with (s_1, z') being the next choice point. Let the set of predicates used to determine the trajectory in terms of active machines and machine states from z.`Top()` to z'.`Top()` be $\mathcal{P} = \{P_1, P_2, \dots\}$. Let the value of $\mathcal{P}$ given state s be $\mathcal{P}(s) = \{P_1(s), P_2(s), \dots\}$. It can be concluded that the transition trajectory induced by $\mathcal{P}$ depends only on $\mathcal{P}(s_1)$, after choice c is made at choice point (s_1, z). On the other hand, if the set of predicates $\mathcal{P}$ over state s_2 ($s_2 \neq s_1$) has the same value as of state s_1, namely $\mathcal{P}(s_2) = \mathcal{P}(s_1)$, and the same choice c is made at choice point (s_2, z), then the followed transition trajectory before reaching the next choice point must also be the same as of $\langle s_1, z, c \rangle$. In other words, $\langle s_2, z, c \rangle$ leads to an internal transition such that $\rho(s_1, z, c) = \rho(s_2, z, c)$.

Thus, the rule of internal transition $\langle s_1, z, c, z' \rangle$ can be equivalently stored and retrieved as $\langle \mathcal{P}, \mathcal{P}(s_1), z, c, z' \rangle$, which automatically applies to $\langle s_2, z, c, z' \rangle$, if $\mathcal{P}(s_2) = \mathcal{P}(s_1)$. Here, z' is the stack of the next choice point such that $z' = \rho(s_1, z, c) = \rho(s_2, z, c)$. The size of the joint space of encountered predicates and their values is determined by the HAM itself, which is typically much smaller than the size of the state space. For example, for a problem with continuous state space (such as the RoboCup Keepaway domain we considered), this joint space is still limited. In summary, we can have an efficient way of storing and retrieving the rules of internal transition by keeping track of the predicates evaluated between two choice points.

3.2 The HAMQ-INT Algorithm

The idea of HAMQ-INT is to identify and take advantage of internal transitions within a HAM. For this purpose, HAMQ-INT automatically keeps track of the predicates that are evaluated between two choice points, stores the discovered rules of internal transition based on predicates and the corresponding values, and uses the learned rules to shortcircuit the computation of Q values whenever it is possible. To detect internal transitions, a global environment time t is maintained. It is incremented by one only when there is an action executed in the environment. When the agent enters a choice point (s', z') after having made a choice c at choice point (s, z), and finds that t is not incremented since the previous choice point, it must be the case that $s' = s$ and $\langle s, z, c \rangle$ leads to an internal transition. Let $\mathcal{P}$ be the set of predicates that have been evaluated between these two choice points. Then a new rule of internal transition $\langle \mathcal{P}, \mathcal{P}(s), z, c, z' \rangle$ is found. The agent can conclude that for any state x, if $\mathcal{P}(x) = \mathcal{P}(s)$, then $\langle x, z, c \rangle$ leads to an internal transition as well. In the implementation, the agent uses a hash table ρ to store the learned rules, such that $\rho[\mathcal{P}, \mathcal{P}(s), z, c] = z'$, if $\langle \mathcal{P}, \mathcal{P}(s), z, c, z' \rangle$ is a rule of internal transition. One thing to note is that, because z' is deterministically conditioned on $\langle \mathcal{P}, \mathcal{P}(s), z, c \rangle$ for an internal transition, the value of $\rho[\mathcal{P}, \mathcal{P}(s), z, c]$ will not be changed after it has been updated for the first time.

When the agent needs to evaluate a Q function, say $Q(s, z, c)$, and finds that $\langle s, z, c \rangle$ leads to an internal transition according to the current learned rules, Equation 1 is used to decompose $Q(s, z, c)$ into the Q values of the next choice points, which are evaluated recursively in the same way, essentially leading to a tree of exact Bellman backups. In fact,



(a) (b)

Figure 2: The Taxi domain (a) and the experimental result (b).

```
Root (s : environment state) :
while not s.Terminate () do
    m ← Choose_1 (Get, Put)
    s ← Run (m, s)
return s

Get (s : environment state) :
s ← Navigate (s, s.Passenger ())
s ← Execute (Pickup, s)
return s

Put (s : environment state) :
s ← Navigate (s, s.Destination ())
s ← Execute (Putdown, s)
return s

Navigate (s : environment state, w : target) :
while s.Taxi () ≠ w do
    m ← Choose_2 (North, East, South, West)
    n ← Choose_3 (1, 2)
    for i ∈ [1, n] do
        s ← Execute (m, s)
return s
```

Figure 3: The HAM in pseudo-code for Taxi.

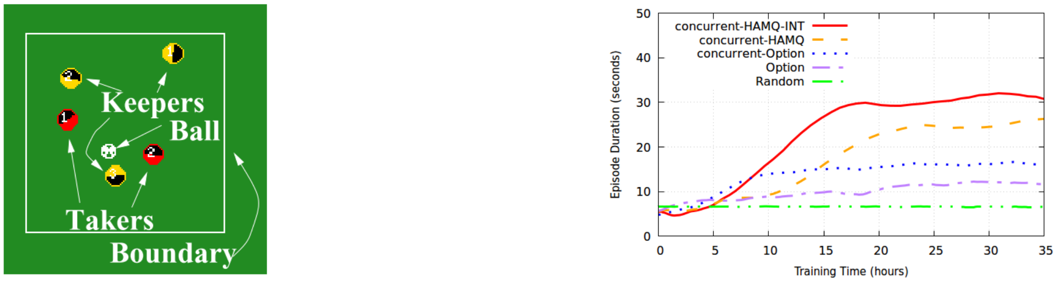
only the terminal Q values of this tree needs to be learned, enabling efficient learning for the agent. Figure 1 gives the pseudo-code of the HAMQ-INT algorithm. Here, the `QTable` function returns the stored Q value as request. It can be implemented in either tabular or function approximation ways. The `Q` function evaluates the Q value of (s, z, c) tuple. It first checks whether (s, z, c) subjects to any learned internal transition rule. This is done by checking whether there exists an encountered set of predicates $\mathcal{P}$, such that $\langle \mathcal{P}, \mathcal{P}(s), z, c \rangle \in \rho.$`Keys ()`. The uniqueness of transition trajectory for an internal transition ensures that there will be at most one such $\mathcal{P}$. If there is such $\mathcal{P}$, `Q` uses the retrieved rule to recursively decompose the requested Q value according to Equation 1; otherwise, it simply returns the stored Q value.

The `QUpdate` function performs the SMDP Q update. It is called once the agent enters a new choice point. The caller has to keep track of the current state s', the current stack z', the evaluated predicates $\mathcal{P}$ on state since the previous choice point and the cumulative reward r in-between. If the current time t' equals to the time t of the previous choice point, it must be the case that $\langle s, z, c, 0, s, z' \rangle$ is an internal transition. Thus, a new rule $\langle \mathcal{P}, \mathcal{P}(s), z, c \rangle$ is learned, and the ρ table is updated accordingly. If $t' \neq t$, meaning there are some actions executed in the environment, it simply performs the Q update. Finally, it uses the current (t', s', z') tuple to update the (global) previous (t, s, z) tuple, so the function will be prepared for the next call.

4 Experiments

4.1 The Taxi Domain

On the Taxi domain, a taxi navigates in a grid map to pick up and deliver a passenger [Dietterich1999]. There are 6 primitive actions for the taxi: a) 4 navigation actions: North, West, South and East; b) the Pickup action; and c) the Putdown action. Each navigation action has probability 0.2 of moving into perpendicular directions. At each time step, unsuccessful Pickup and Putdown have a reward of -10, successful Putdown has a reward of 20, and all other actions have a reward of -1. Figure 3 shows the HAM written in pseudo-code for this experiment. It is worth noting that **Choose**$_3$ is used for encouraging the discovery of temporally-extended action and creating more opportunities of internal transitions. The internal transition happens after a choice is made within the **Root** machine. For example, when a **Get** machine is selected at the choice point **Choose**$_1$ within the **Root** machine, the next choice point must be **Choose**$_2$ within the **Navigate** machine. We compare HAMQ-INT with HAMQ, MAXQ-0 and MAXQ-Q algorithms. The standard task graph for Taxi is used for MAXQ-0 and MAXQ-Q algorithms. MAXQ-Q is additionally encoded with a pseudo-reward function for the Navigation sub-task, which gives a reward of 1 when successfully terminated. Comparing with MAXQ algorithms, one



(a) (b)

Figure 4: The RoboCup Keepaway domain (a) and the experimental result (b).

```
Keeper ($s$ : environment state):
while not $s$.Terminate() do
    if $s$.BallKickable() then
        $m \leftarrow$ Choose$_1$ (Pass, Hold)
        $s \leftarrow$ Run ($m$, $s$)
    else if $s$.FastestToBall() then
        $s \leftarrow$ Intercept ($s$)
    else
        $m \leftarrow$ Choose$_2$ (Stay, Move)
        $s \leftarrow$ Run ($m$, $s$)
return $s$

Move ($s$ : environment state):
$d \leftarrow$ Choose$_5$ ($0^\circ, 90^\circ, 180^\circ, 270^\circ$)
$v \leftarrow$ Choose$_6$ (Normal, Fast)
$i \leftarrow s$.TmControlBall()
while $i = s$.TmControlBall() do
    $v \leftarrow$ Run (Move, $d$, $v$)
return $s$

Pass ($s$ : environment state):
$k \leftarrow$ Choose$_3$ ($1, 2, \ldots$)
$v \leftarrow$ Choose$_4$ (Normal, Fast)
while $s$.BallKickable() do
    $s \leftarrow$ Run (Pass, $k$, $v$)
return $s$

Hold ($s$ : environment state):
$s \leftarrow$ Run (Hold)
return $s$

Intercept ($s$ : environment state):
$s \leftarrow$ Run (Intercept)
return $s$

Stay ($s$ : environment state):
$i \leftarrow s$.TmControlBall()
while $i = s$.TmControlBall() do
    $s \leftarrow$ Run (Stay)
return $s$
```

Figure 5: The HAM for RoboCup Keepaway.

advantage of HAM is that the `Get` and `Put` machines are encoded with the right order of calling other machines, while MAXQ algorithms have to learn this order by themselves. Figure 2b shows the experimental result averaged over 400 runs. It can be seen from the result that MAXQ-Q outperforms MAXQ-0 as expected, HAMQ outperforms MAXQ-Q at the early stage of learning, and HAMQ-INT outperforms HAMQ significantly.

4.2 The RoboCup Keepaway Domain

The RoboCup Keepaway problem is a sub-task of RoboCup soccer simulation 2D challenge [Stone *et al.*2005, Bai *et al.*2015]. In Keepaway, a team of keepers tries to maintain the ball possession within a limited field, while a team of takers tries to take the ball. Figure 4a shows an instance of Keepaway with 3 keepers and 2 takers. The system has continuous state and action spaces. A state encodes positions and velocities for the ball and all players. At each time step (within 100 ms), a player can execute a parametrized primitive action, such as turn($angle$), dash($power$) or kick($power$, $angle$), where the turn action changes the body angle of the player, the dash action gives an acceleration to the player, and the kick action gives an acceleration to the ball if the ball is within the maximal kickable area of the player. All primitive actions are exposed to noises. Each episode begins with the ball and all players at fixed positions, and ends if any taker kicks the ball, or the ball is out of the field. The cumulative reward for the keepers is the total number of time steps for an episode. Instead of learning to select between primitive actions, the players are provided with a set of programmed options including: 1) **Stay**() remaining stationary at the current position; 2) **Move**(d, v) dashing towards direction d with speed v; 3) **Intercept**() intercepting the ball; 4) **Pass**(k, v) passing the ball to teammate k with speed v; and 5) **Hold**() remaining stationary while keeping the ball kickable. In our experiments, the taker is executing a fixed policy. It holds the ball if the ball is kickable, otherwise it intercepts the ball. This policy is commonly used in the literature. The goal is then to learn a best-response policy for the keepers given fixed takers. We develop a HAM policy from the perspective of a single keeper, and run multiple instances of this HAM concurrently for each keeper to form a joint policy for all keepers. To run multiple HAMs concurrently, they have to be synchronized, such that if any machine is at its **choose** state, the other machines have to wait; if multiple machines are at their **choose** states, a joint choice is made instead of independent choice for each machine. For this purpose, players have to share their learned value functions and the selected joint choice. A joint Q update is developed to learn the joint choice selection policy as an optimal completion of the resulting joint HAM. Figure 5 outlines the HAM written in pseudo-code for a single keeper. Here, `Keeper` is the root machine. The `Run` macro runs a machine or an option with specified parameters. `BallKickable`, `FastestToBall`, `TmControlBall` are predicates used to determine the transition inside a machine. It is worth noting that the `Move` machine only considers 4 directions, with direction 0° being the direction towards the ball, and so on. There are many internal transitions within this single HAM. For example, when the `Pass` machine is selected at the choice point `Choose`$_1$ of the `Keeper` machine, the next 2 consecutive choice points must be `Choose`$_3$ and `Choose`$_4$ within the `Pass` machine. When multiple HAMs are executing concurrently, there are even more internal transitions in the resulting joint HAM. For example, in a scenario of the 3 vs. 2 Keepaway game, where only keeper 1 can kick the ball, suppose the joint machine state is [`Choose`$_1$, `Choose`$_2$, `Choose`$_2$] with each element being the machine state of a HAM. If the joint choice made is [`Pass`, `Move`, `Stay`], then the next 2 consecutive machine states must be [`Choose`$_3$, `Choose`$_5$, `Stay`] and [`Choose`$_4$, `Choose`$_6$, `Stay`] following the joint HAM.

We compare concurrent-HAMQ-INT, concurrent-HAMQ, concurrent-Option, Option and Random algorithms. The Option algorithm is adopted from [Stone *et al.*2005], where the agent learns an option-selection policy over **Hold**() and **Pass**(k, v) options if it can kick the ball, otherwise it follows a fixed policy: if it is the fastest one to intercept the ball, it intercepts; otherwise, it follows a **GetOpen**() option. The **GetOpen**() option, which enables the agent to move to an open area in the field, is manually programmed beforehand. In the original Option learning algorithm, each agent learns independently. We argue that this setting is problematic, since it actually incorrectly assumes that other keepers are stationary. We extend Option to concurrent-Option, by sharing the learned value functions and the option selected. The HAM algorithms are not provided with the **GetOpen**() option. Instead, they have to learn their own versions of **GetOpen**() by selecting from `Stay` and `Move` machines. The SARSA-learning rule with a linear function approximator (namely *tile coding*) is used to implement the SMDP Q update for all learning algorithms. The Random algorithm is a non-learning version of Option, which selects available options randomly as a baseline. Figure 4b shows the experiment result on a 3 vs. 2 instance of RoboCup Keepaway averaged using a moving window with size of 1000 episodes. It can be seen from the result that concurrent-Option outperforms Option significantly, concurrent-HAMQ outperforms concurrent-Option after about 15 hours of training, and concurrent-HAMQ-INT has the best performance. Short videos showing the initial and converged policies of HAMQ-INT can be found at 1 and 2 links.

5 Conclusion

In this paper, we present a novel HAMQ-INT algorithm which automatically discovers and takes advantage internal transitions within a HAM for efficient learning. We empirically confirm that HAMQ-INT outperforms the state of the art significantly on the benchmark Taxi domain and a much more complex RoboCup Keepaway domain. The way we taking advantage of internal transitions within a HAM can be seen as leveraging some prior knowledge on the transition model of a reinforcement learning problem, which happens to have some deterministic transitions. In future work, we would like to extend this idea to more general reinforcement learning problems, where models are partially known in advance.

References

[Andre and Russell2002] David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *Proceedings of the 8th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence*, pages 119–125, 2002.

[Bai *et al.*2015] Aijun Bai, Feng Wu, and Xiaoping Chen. Online planning for large Markov decision processes with hierarchical decomposition. *ACM Transactions on Intelligent Systems and Technology*, 6(4):45, 2015.

[Barto and Mahadevan2003] A.G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13:341–379, 2003.

[Dietterich1999] Thomas G Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Machine Learning Research*, 13(1):63, May 1999.

[Marthi *et al.*2005] Bhaskara Marthi, Stuart J Russell, David Latham, and Carlos Guestrin. Concurrent hierarchical reinforcement learning. In *IJCAI*, pages 779–785, 2005.

[Parr and Russell1998] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, volume 10, 1998.

[Stone *et al.*2005] P. Stone, R.S. Sutton, and G. Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[Sutton *et al.*1999] R.S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

Deep Reinforcement Learning with Model Learning and Monte Carlo Tree Search in Minecraft

Stephan Alaniz
Department of Electrical Engineering and Computer Science
Technische Universität Berlin
Berlin, Germany
`stephan.alaniz@gmail.com`

Abstract

Deep reinforcement learning has been successfully applied to several visual-input tasks using model-free methods. In this paper, we propose a model-based approach that combines learning a DNN-based transition model with Monte Carlo tree search to solve a block-placing task in Minecraft. Our learned transition model predicts the next frame and the rewards one step ahead given the last four frames of the agent's first-person-view image and the current action. Then a Monte Carlo tree search algorithm uses this model to plan the best sequence of actions for the agent to perform. On the proposed task in Minecraft, our model-based approach reaches the performance comparable to the Deep Q-Network's, but learns faster and, thus, is more training sample efficient.

Keywords: Reinforcement Learning, Model-Based Reinforcement Learning, Deep Learning, Model Learning, Monte Carlo Tree Search

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Stefan Uhlich for his continuous support, patience, and immense knowledge that helped me a lot during this study. My thanks and appreciation also go to my colleague Anna Konobelkina for insightful comments on the paper as well as to Sony Europe Limited for providing the resources for this project.

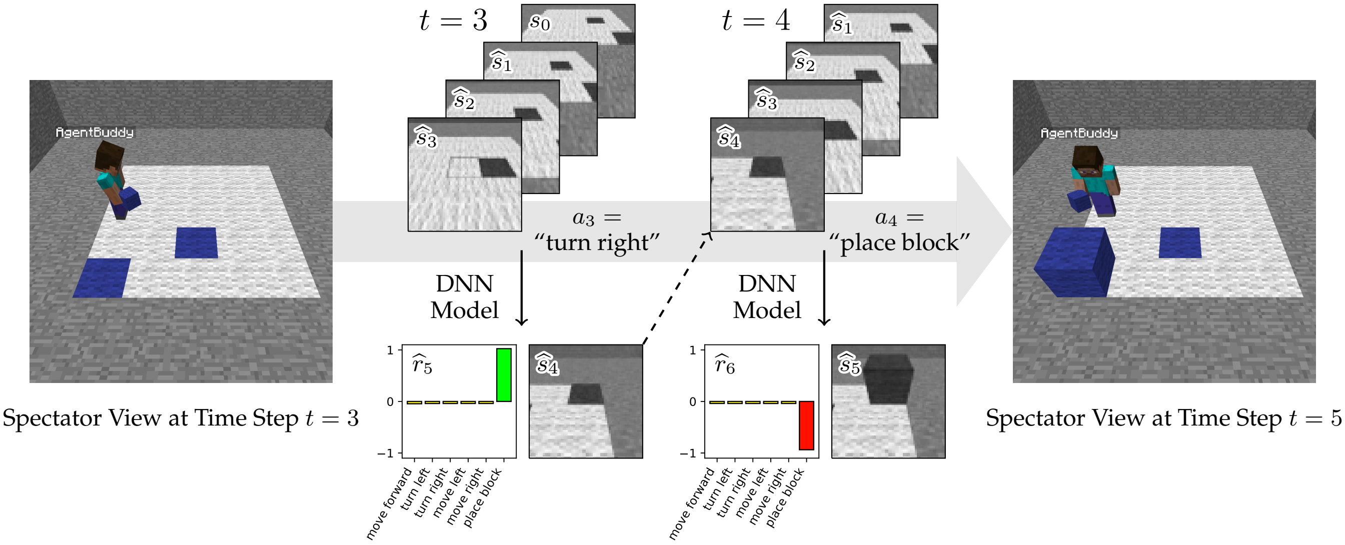


Figure 1: Transition Model for the Block-Placing Task in Minecraft

1 Introduction

In deep reinforcement learning, visual-input tasks (i.e., tasks where the observation from the environment comes in the form of videos or pictures) are oftentimes used to evaluate algorithms: Minecraft or various Atari games appear quite challenging for agents to solve [Oh+16], [Mni+15]. When applied to these tasks, model-free reinforcement learning shows noticeably good results: e.g., a Deep Q-Network (DQN) agent approaches human-level game-playing performance [Mni+15] or the Asynchronous Advantage Actor-Critic algorithm outperforms the known methods in half the training time [Mni+16]. These achievements, however, do not cancel the fact that generally, model-free methods are considered "statistically less efficient" in comparison to model-based ones: model-free approaches do not employ the information about the environment directly whereas model-based solutions do [DN08].

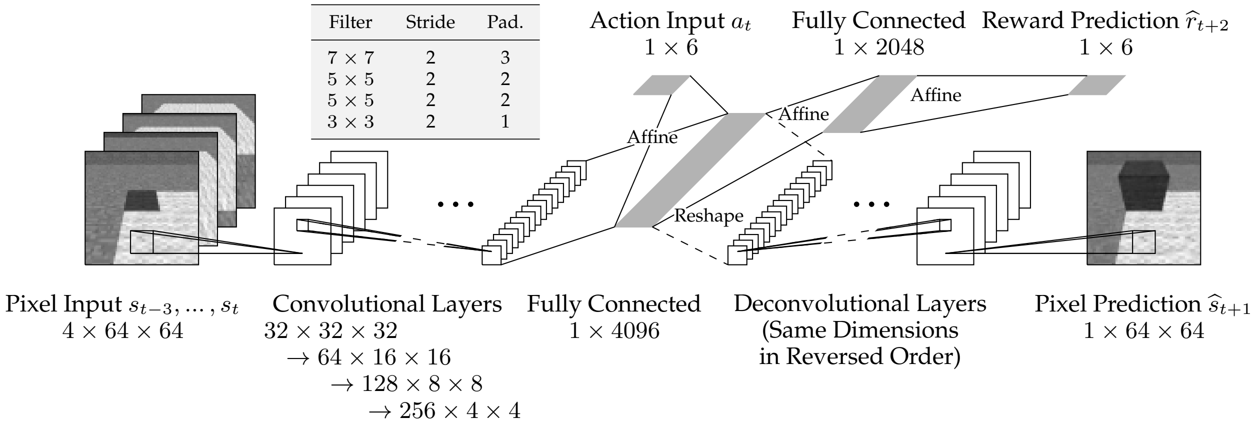
Working with a known environment model has its benefits: changes of the environment state can be foreseen, therefore, planning the future becomes less complicated. At the same time, developing algorithms with no known environment model available at the start is more demanding yet promising: less training data is required than for model-free approaches and agents can utilize planning algorithms. The research has been progressing in this direction: e.g., a model-based agent surpassed the DQN's results by using the Atari games' true state for modelling [Guo+14], and constructing transition models with video-frames prediction was proposed [Oh+15].

These ideas paved the way for the following question: is it feasible to apply planning algorithms on a learned model of the environment that is only partially observable, such as in a Minecraft building task? To investigate this question, we developed a method that not only predicts future frames of a visual task but also calculates the possible rewards for the agent's actions. Our model-based approach merges model learning through deep-neural-network training with Monte Carlo tree search, and demonstrates results competitive with those of DQN's, when tested on a block-placing task in Minecraft.

2 Block-Placing Task in Minecraft

To evaluate the performance of the suggested approach as well as to compare it with model-free methods, namely, DQN, a block-placing task was designed: it makes use of the Malmo framework and is built inside the Minecraft game world [Joh+16].

At the beginning of the game, the agent is positioned to the wall of the playing "room". There is a 5×5 playing field in the center of this room. The field is white with each tile having a 0.1 probability to become colored at the start. Colored tiles indicate the location for the agent to place a block. The goal of the game is to cover all the colored tiles with blocks in 30 actions maximum. Five position-changing actions are allowed: moving forward by one tile, turning left or right by 90°, and moving sideways to the left or right by one tile. When the agent focuses on a tile, it can place the block with the 6th action. For each action, the agent receives a feedback: every correctly placed block brings a +1 reward, an erroneously put block causes a -1 punishment, and any action costs the agent a -0.04 penalty (this reward signal is introduced to stimulate the agent to solve the task with the minimum time required). To evaluate the environment, the agent is provided with a pre-processed (grayscaled and downsampled to 64×64) first-person-view picture of the current state. The task is



Filter	Stride	Pad.
7×7	2	3
5×5	2	2
5×5	2	2
3×3	2	1

Figure 2: Architecture of the Transition Model

deterministic and discrete in its action and state space. The challenge lies in the partial observability of the environment with already placed blocks further obscuring the agent's view. It is equally important to place blocks systematically to not obstruct the agent's pathway. An example of the task is depicted in Fig. 1 (left) and a short demonstration is available at https://youtu.be/AQlBaq34DpA.

3 Model Learning

To learn the transition model, a deep convolutional neural network is used. The network takes the last four frames $s_{t-3}, \dots, s_t$, and an action a_t as an input, and predicts the following frame $\widehat{s}_{t+1}$. Additionally, it predicts the rewards for all the transitions following the predicted frame $\widehat{s}_{t+1}$, one for each action a_{t+1}. Predicting the rewards one step ahead makes the application of search-based algorithms more efficient as no additional simulation is required to explore rewards from transitions to neighboring states. This method, however, fails to predict the reward following the very first state. To address this issue, a "noop" action, predicting the reward of the current state, is introduced.

The network takes four 64×64-sized input frames and uses four convolutional layers, each followed by a rectifier linear unit (ReLU), to encode the input information into a vector of the size of 4096. This vector is concatenated with the one-hot encoded action input, where the "noop" action is represented by a vector of all zeros, and then linearly transformed with a fully connected layer of the size of 4096, again followed by ReLU. The resulting embedded vector is used for both, the reward prediction and the frame prediction, in the last part of the network. For the frame prediction, four deconvolutional layers are used with ReLUs in between and a Sigmoid at the end. The dimensions of these layers are equivalent to the convolutional layers in the first part in reversed order. The reward prediction is done by applying two fully connected linear layers of the sizes of 2048 and 6, respectively. ReLU is used in between the two layers, but not at the final output where we predict the rewards. The architecture of the neural network is illustrated in Fig. 2.

Training of the network is done with the help of experience replay [Mni+15] to re-use and de-correlate the training data. Mini-batches of the size of 32 are sampled for training and RMSProp [TH12] is used to update the weights. Both, the frame prediction and the reward prediction, are trained with a mean squared error (MSE) loss. On each mini-batch update, the gradients of the frame prediction loss are backpropagated completely through the network while the gradients of the reward prediction loss are backpropagated only two layers until the embedded vector shared with the frame prediction is encountered. This procedure ensures that the network uses its full capacity to improve the prediction of the next state and the reward prediction is independently trained on the embedded feature vector from the last shared intermediate layer. Due to the network's structure, the shared layer may only contain the necessary information to construct the prediction of the next frame and no further past information. For the block-placing task , one frame suffices to predict the reward. For different tasks, using the previous layer with the action input is worth considering (cf. Fig. 2).

4 Monte Carlo Tree Search

Finding actions with maximum future reward is done with the help of a UCT-based strategy, Monte Carlo tree search (MCTS) [Cou06; KS06], and the learned model described in Sec. 3. The input for the next step is based on the frame prediction of the model. This procedure can be repeated several times to roll out the model to future states. One tree-search trajectory is rolled out until a maximum depth is reached. During one rollout, all the rewards along the trajectory are predicted as well as the rewards for neighboring states due to the network's structure. As the outputs of the neural network are deterministic, each state is evaluated only once but can still be visited multiple times since many paths go

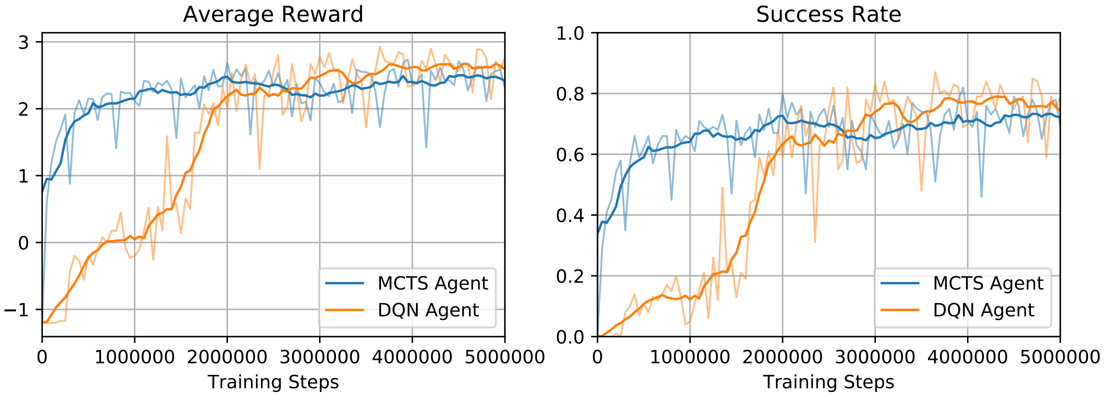


Figure 3: MCTS Agent vs. DQN Agent: Average Reward and Success Rate on 100 Block-Placing Tasks in Minecraft. One training step = one mini-batch update; raw values (transparent lines, evaluated every 50K steps) smoothed with moving average (solid lines, 200K steps to each side).

through the same states. The decision about which action to follow for each state during a tree-search trajectory is made based on the UCT-measure $UCT_s = v_s + k\sqrt{(\ln n_p)/n_s}$. The action with the maximum UCT value is chosen greedily, where v_s is the maximum discounted future reward of the state s so far, n_s is the number of visits of state s, and n_p is the number of visits of its parent. The hyperparameter k controls the trade-off of exploration and exploitation, where a higher k translates to more exploration. If a state has not been visited yet, it will be preferred over the already visited ones. In case several states under consideration have not been visited yet, the one with the highest immediate reward is chosen. The reward is given by the neural network model that predicts rewards one step ahead. Whenever a path of higher maximum value is encountered during a search trajectory, the path's value is propagated to the tree's root node updating every node with the new maximum reward value.

During the evaluation of the task in Minecraft, the agent performs MCTS for each action decision it has to make. The agent is given a fixed number of trajectories to roll out and decides for the action of maximum future discounted reward to take as the next action. Subsequently, the agent receives a new ground-truth input frame from the environment for the last step and updates the root of the search tree with the new state information. The next step is again chosen by applying MCTS beginning from the new state. Instead of starting from scratch, the UCT value is calculated with the maximum future reward value of the previous turn. This way, the tree-search results are carried over to following steps and trajectories with maximum future reward are updated first with the new input.

5 Experiments and Results

The MCTS agent uses a maximum depth of ten for its rollouts and explores 100 trajectories before deciding for the next action. Over the course of solving a single task, the MCTS agent explores 3000 trajectories at most since the task is limited to 30 actions. Trajectories, however, may be repeated with new input frames arriving after every action decision. The hyperparameter k of UCT is set to eight. Compared to other domains, this value is rather large. The reason behind this decision is as follows: our exploitation measure (maximum future reward) is not limited in range, and the number of visits remain low since each node is evaluated only once. We use the original DQN architecture [Mni+15] to train a DQN agent for the block-placing task. Restricting the input to four frames for our method grounds in the same number of frames used in the DQN's structure to provide both approaches with equal input information. In Fig. 3, the results for task-solving success rate and the average reward are presented. For both agents, MCTS and DQN, the two performance measures were evaluated over the course of training steps, where each step corresponds to a mini-batch update. As a test set, 100 block-placing tasks were employed. At the end of the training, both agents appear to level out at roughly the same scores. The DQN agent overestimates the Q-values in the first 1.7 million training steps. Although reducing the learning rate helped weakening this effect, it slowed down the training process. This is a known problem with Q-learning.

As for the model-based approach, it can quickly learn a meaningful model that can achieve good results with MCTS. This suggests that learning the transition function is an easier task than learning Q-values for this block-placing problem. The main pitfall of the model-based approach lies in accumulating errors for rollouts that reach many steps ahead. In this particular block-placing task, MCTS trajectories are only 10 steps deep and Minecraft frames are rather structured, hence, the accumulating error problem is not that prominent. The slight increase in variance of the reward predictions of future steps is alleviated by using a discount rate of 0.95 for the MCTS agent as compared to a discount rate of 0.99 used by DQN. This value was found empirically to provide the best results.

	Average Reward			Success Rate		
Agent	1M steps	2.5M steps	5M steps	1M steps	2.5M steps	5M steps
MCTS	**2.15**	**2.36**	2.42	**0.64**	**0.70**	0.72
MCTS, no 1-ahead reward	0.68	0.84	0.87	0.27	0.30	0.31
DQN	0.05	2.22	**2.59**	0.12	0.64	**0.74**

Table 1: Results of Different Algorithms for the Block-Placing Task in Minecraft (cf. smoothed values in Fig. 3).

Table 1 demonstrates the smoothed scores for both agents after 1, 2.5, and 5 million training steps. After 1 million steps, the model-based approach can already solve a considerate amount of block-placing tasks whereas the DQN agent has not learned a reasonable policy yet. For the DQN agent to catch up with the MCTS agent, it needs 1.5 million additional training steps, what underlines the model-based approach's data efficiency. In the end, DQN beats the MCTS agent only by a small margin (74% vs. 72% success rate). Additionally, the table includes the MCTS agent's scores if the one-step-ahead prediction of the reward is not employed. During the tree search, this agent chooses a random action of unexplored future states instead of greedily choosing the action with maximum immediate reward. For the block-placing task, using one-step-ahead predicted rewards doubles the score across training steps, i.e., scores comparable with the DQN agent become achievable with only 100 trajectories.

6 Conclusion

In this paper, we explored the idea of creating a model-based reinforcement learning agent that could perform competitively with model-free methods, DQN, in particular. To implement such an agent, a synthesis of learning a transition model with a deep neural network and MCTS was developed. Our tests on a block-placing task in Minecraft show that learning a meaningful transition model requires considerably less training data than learning Q-values of comparable scores with DQN. As the MCTS agent uses a tree search for finding the best action, it takes longer to perform one action in comparison with DQN. Therefore, our approach is interesting for cases where obtaining training samples from the environment is costly. The nature of the block-placing task justifies the greedy choice of immediate rewards and, hence, application of the one-step-ahead prediction significantly improves the score performance. The transition model suffers from the limited information of the last four input frames: if past information becomes unavailable (e.g., no longer visible in the last four frames) the model makes incorrect predictions about the environment leading to suboptimal actions. Further research in using a recurrent neural network could help eliminating this issue.

References

[Cou06] Rémi Coulom. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". In: *Proceedings of the 5th International Conference on Computers and Games*. CG'06. Turin, Italy: Springer-Verlag, 2006, pp. 72–83.

[DN08] Peter Dayan and Yael Niv. "Reinforcement Learning: The Good, The Bad and The Ugly". In: *Current Opinion in Neurobiology* 18.2 (Apr. 2008), pp. 185–196.

[Guo+14] Xiaoxiao Guo et al. "Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 3338–3346.

[Joh+16] Matthew Johnson et al. "The Malmo Platform for Artificial Intelligence Experimentation". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*. 2016, pp. 4246–4247.

[KS06] Levente Kocsis and Csaba Szepesvári. "Bandit Based Monte-Carlo Planning". In: *Proceedings of the 17th European Conference on Machine Learning*. ECML'06. Berlin, Germany: Springer-Verlag, 2006, pp. 282–293.

[Mni+15] Volodymyr Mnih et al. "Human-Level Control through Deep Reinforcement Learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533.

[Mni+16] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016*. 2016, pp. 1928–1937.

[Oh+15] Junhyuk Oh et al. "Action-Conditional Video Prediction Using Deep Networks in Atari Games". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 2845–2853.

[Oh+16] Junhyuk Oh et al. "Control of Memory, Active Perception, and Action in Minecraft". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016*. 2016, pp. 2790–2799.

[TH12] T. Tieleman and G. Hinton. *Lecture 6.5–RmsProp: Divide the Gradient by a Running Average of its Recent Magnitude*. COURSERA: Neural Networks for Machine Learning. 2012.

Using Advice in Model-Based Reinforcement Learning

Rodrigo Toro Icarte
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
`rntoro@cs.toronto.edu`

Toryn Q. Klassen
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
`toryn@cs.toronto.edu`

Richard Valenzano
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
`rvalenzano@cs.toronto.edu`

Sheila A. McIlraith
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
`sheila@cs.toronto.edu`

Abstract

When a human is mastering a new task, they are usually not limited to exploring the environment, but also avail themselves of *advice* from other people. In this paper, we consider the use of advice expressed in a formal language to guide exploration in a model-based reinforcement learning algorithm. In contrast to constraints, which can eliminate optimal policies if they are not sound, advice is merely a recommendation about how to act that may be of variable quality or incomplete. To provide advice, we use Linear Temporal Logic (LTL), which was originally proposed for the verification of properties of reactive systems. In particular, LTL formulas can be used to provide advice to an agent about how they should behave over time by defining a temporal ordering of state properties that the agent is recommended to avoid or achieve. LTL thus represents an alternative to existing methods for providing advice to a reinforcement learning agent, which explicitly suggest an action or set of actions to use in each state.

We also identify how advice can be incorporated into a model-based reinforcement learning algorithm by describing a variant of R-MAX which uses an LTL formula describing advice to guide exploration. This variant is guaranteed to converge to an optimal policy in deterministic settings and to a near-optimal policy in non-deterministic environments, regardless of the quality of the given advice. Experimental results with this version of R-MAX on deterministic grid world MDPs demonstrate the potential for good advice to significantly reduce the number of training steps needed to learn strong policies, while still maintaining robustness in the face of incomplete or misleading advice.

Keywords: Markov Decision Process, Reinforcement Learning, Model-Based Learning, Linear Temporal Logic, Advice.

Acknowledgements

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC). The first author also gratefully acknowledges funding from CONICYT (Becas Chile).

1 Introduction

Reinforcement Learning (RL) methods can often be used to build intelligent agents that learn how to maximize long-term cumulative reward through interaction with the environment. Doing so generally requires extensive exploration of the environment, which can be infeasible in real-world environments in which exploration can be unsafe or requires costly resources. Even when there is access to a simulator and exploration is safe, the amount of interaction needed to find a reasonable policy may be prohibitively expensive.

In this paper we investigate the use of *advice* as a means of guiding exploration. Indeed, when humans try to master a new task, they certainly learn through exploration, but they also avail themselves of linguistically expressed advice from other humans. Here we take "advice" to be recommendations regarding behaviour that may describe suboptimal ways of doing things, may not be universally applicable, or may even contain errors. However, even in these cases people often extract value and we aim to have RL agents do likewise. We use advice to guide the exploration of an RL agent during its learning process so that it more quickly finds useful ways of acting. Advice can be contrasted with hard constraints, the imposition of which potentially eliminates optimal policies if the constraints are incorrect or suboptimal.

We use the language of Linear Temporal Logic (LTL) for providing advice. The advice vocabulary is drawn from state features, together with the linguistically inspired temporal modalities of LTL (*e.g.*, *"Turn out the lights before you leave the office"* or *"Always avoid potholes in the road"*). We propose a variant of the standard model-based RL algorithm *R-MAX* that adds the ability to use given advice to guide exploration. Experimental results on randomly generated grid worlds demonstrate that our approach can effectively use good advice to reduce the number of training steps needed to learn a strong policy. Our approach is robust in the face of incomplete or misleading advice, as we can guarantee convergence to an optimal policy in deterministic settings and convergence to a near optimal policy in non-deterministic environments.

Related Work: Maclin and Shavlik [6] proposed the use of advice in RL via a procedural programming language that supports recommendation of actions using `If-Then` rules and loops. Krening *et al.* [5] grounded natural language advice into a list of *(object,action)* recommendations (or warnings), to encourage (or discourage) an agent to perform an action when it interacts with particular classes of objects. Neither of these methods incorporate advice into a model-based RL algorithm. Moreover, our LTL advice language is less prescriptive: it allows a user to provide a rich temporal ordering of state properties that the agent may wish to avoid or achieve, as opposed to explicitly specifying the actions to take in a given state. This allows for alternative forms of advice to those considered by earlier works.

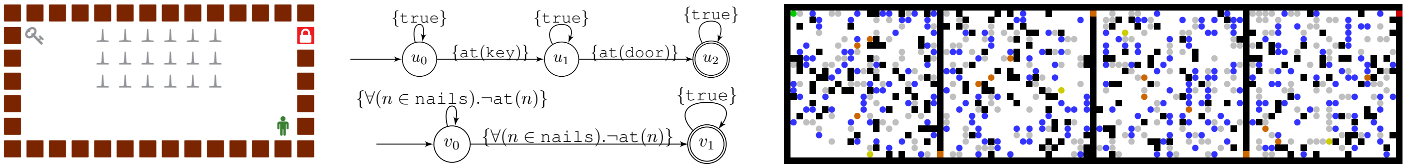
2 Preliminaries

Example environment: Consider a grid world in which the agent starts at some initial location. At various locations there are doors, keys, walls, and nails. The agent can move deterministically in the four cardinal directions, unless there is a wall or locked door in the way. The agent can only enter a location with a door when it has a key, after which point the door and key disappear (*i.e.* the door remains open). The agent automatically picks up a key whenever it visits a location with a key. The agent receives a reward of -1 for every action, unless it enters a location with nails (reward of -10) or reaches the red door with a key (reward of +1000, and the episode ends). Figure 1a, which we use as a running example below, depicts an instance of this domain, the "nail room", in which there is a single door and it is red.

In this environment, we may wish to advise the agent to avoid the nails, or to get the key before going to the door. In order to provide advice to an arbitrary RL agent, we must have a vocabulary from which the advice is constructed. For this purpose, we define a *signature* as a tuple $\Sigma = \langle \Omega, C, \mathrm{arity} \rangle$ where Ω is a set of predicate symbols, C is a set of constant symbols, and $\mathrm{arity} : \Omega \to \mathbb{N}$ assigns an arity to each predicate. For example, in the grid-world environment, we use a signature with only a single predicate called `at` (*i.e.* $\Omega = \{\mathtt{at}\}$ and $\mathrm{arity}(\mathtt{at}) = 1$), where $\mathtt{at}(c)$ states that the agent is at the same location as c. Each object in the domain will be represented with a single constant in C (*i.e.* $\mathrm{key1}, \mathrm{door1}, \ldots$). Intuitively, we use this signature to reference different elements about states in the environment when providing advice.

We assume that the set C is finite, and define $GA(\Sigma) := \{P(c_1, \cdots, c_{\mathrm{arity}(P)}) \mid P \in \Omega, c_i \in C\}$. That is, $GA(\Sigma)$ is the set of all *ground atoms* of the first order language with signature Σ. A *ground literal* is either a ground atom or the negation of a ground atom, so $\mathsf{lit}(\Sigma) := GA(\Sigma) \cup \{\neg p : p \in GA(\Sigma)\}$ is the set of ground literals. A *truth assignment* can be given by a set $\tau \subseteq \mathsf{lit}(\Sigma)$ such that for every $a \in GA(\Sigma)$, exactly one of a and $\neg a$ is in τ. Let $T(\Sigma)$ be the set of all truth assignments.

A *Markov Decision Process (MDP)* with a specified initial state is a tuple $\mathcal{M} = \langle S, s_0, A, p, \gamma, \Sigma, L \rangle$ where S is a finite set of *states*, $s_0 \in S$ is the initial state, $A(s)$ is a finite set of *actions* applicable in state $s \in S$, p is a function that specifies *transition probabilities* where $p(s', r|s, a)$ is the probability of transitioning to s' and receiving reward $r \in \mathbb{R}$ if action $a \in A(s)$ is taken in state s, $\gamma \in (0, 1]$ is the *discount factor*, $\Sigma = \langle \Omega, C, \mathrm{arity} \rangle$ is a signature, and $L : S \to T(\Sigma)$ labels each state with a truth assignment. For example, in our grid-world, the labeling function $L(s)$ makes $\mathtt{at}(c)$ true if and only if the location of the agent is equal to the location of c in state s. Note that as $GA(\Sigma)$ is finite, we could equivalently consider a state label to be a vector of *binary features*, with the ith entry being 1 if the ith ground atom holds in that state, and 0 otherwise. Below, we assume that the agent does not know the transition probability function p (as usual in RL).



(a) The nail room, an example (b) Advice example (c) An example randomly generated grid world map

Figure 1: (a) The agent receives reward for going to the locked door after having visited the key. It is penalized for stepping on nails. (b) NFAs corresponding to the LTL formula $\Diamond(\texttt{at(key)} \wedge \bigcirc\Diamond\texttt{at(door)}) \wedge \Box\forall(n \in \texttt{nails}).\neg\texttt{at}(n)$. (c) Random map that contains an agent (●), walls (■), nails (●), holes (●), keys (●), cookies (●), doors (■), and a final door (■).

3 Providing and Utilizing Advice While Learning

In this section, we describe our LTL advice language and how corresponding automata can be used to monitor satisfaction of such advice. We then describe our method for using the automata to guide exploration in a variant of R-MAX.

Linear Temporal Logic: A Language for Providing Advice

Providing advice to an agent requires a language for communicating that advice. Given that RL agents are typically engaged in an extended interaction with the environment, this language must allow us to suggest how the agent should behave over time (*e.g.* *"Get the key and then go to the locked door."*). To this end, we use Linear Temporal Logic (LTL), a modal temporal logic originally proposed for the verification of reactive systems [8], that has subsequently been used to represent temporally extended goals and preferences in planning [4]. Here, we use it as the basis for expressing advice.

Suppose that we have an MDP $\mathcal{M} = \langle S, s_0, A, p, \gamma, \Sigma, L\rangle$ for which we wish to provide advice. The language of LTL contains formulae consisting of propositional symbols, which we take to be the ground atoms of $GA(\Sigma)$ (*e.g.* $\texttt{at(key2)}$), and all formulae that can be constructed from other formulae using the standard set of connectives from propositional logic — namely *and* ($\wedge$), *or* ($\vee$), and *negation* ($\neg$) — and the temporal operators *next* ($\bigcirc$) and *until* (U). From these temporal operators, we can also derive other useful operators such as *always* ($\Box$) and *eventually* ($\Diamond$). We will also make use of *universal* ($\forall$) and *existential* ($\exists$) quantifiers in abbreviating conjunctions and disjunctions. If $T = \{t_1, \ldots, t_k\} \subseteq C$ is a set of constant symbols, then $\forall(x \in T).\varphi(x) := \varphi(t_1) \wedge \cdots \wedge \varphi(t_k)$ and $\exists(x \in T).\varphi(x) := \varphi(t_1) \vee \cdots \vee \varphi(t_k)$.

To provide some intuition about LTL, we consider some possible example advice formulae for the problem in Figure 1a which use the unary operators $\Box$, $\Diamond$, and $\bigcirc$. The formula $\Box\neg\texttt{at}(\text{nail1})$ literally means "always, it is not the case that the agent is at the location of nail1"; used as advice it can be taken to say that "at all times the agent *should* not be at nail1." The formula $\Diamond(\texttt{at}(\text{key}) \wedge \bigcirc(\Diamond\texttt{at}(\text{door})))$ can be understood as "the agent should eventually get to a state where it is at the key and then eventually get to a state where it is at the door." If $\texttt{nails} \subset C$ is the set of objects that are nails, we can use $\Box\forall(n \in \texttt{nails}).\neg\texttt{at}(n)$ to advise that at all times, the agent should not be at a location where there is a nail.

The truth value of an LTL formula φ is determined relative to a sequence $\sigma = \langle s_0, \ldots, s_n\rangle$ of states from $\mathcal{M}$ (*i.e.* the states visited in an episode). The truth of a ground atom at time t is determined by the label of s_t, and the truth values of more complicated formulae are built up according to the formal semantics of LTL (see De Giacomo *et al.* [3] for more detail).

From LTL to Finite State Automata and Tracking Advice Progress

Any LTL formula φ can be converted into a *Nondeterministic Finite State Automaton (NFA)* such that a finite sequence of states σ will be accepted by the NFA if and only if σ satisfies φ [1, 3]. We can represent the NFA as a directed graph with edges labelled by formulae from $\mathcal{L}_\Sigma$, the subset of LTL formulae that does not include temporal operators. Each edge represents a set of NFA transitions, one for each truth assignment satisfying the edge label. Because the NFA is non-deterministic, it may be in multiple states at once. Intuitively, the state(s) that an NFA is in after visiting a sequence of MDP states represent the progress that has been made towards satisfying φ. As such, we use an NFA state set to identify the relevant portion of a given advice formula in any state.

To translate from LTL to automata we use the system developed by Baier and McIlraith [1], which, for computational efficiency, constructs a set $\mathcal{N}$ of small NFAs rather than one potentially very large NFA. $\mathcal{N}$ is considered to accept σ if every NFA in $\mathcal{N}$ accepts σ. For example, Figure 1b shows the two NFAs generated from $\Diamond(\texttt{at(key)} \wedge \bigcirc\Diamond\texttt{at(door)}) \wedge \Box\forall(n \in \texttt{nails}).\neg\texttt{at}(n)$. This advice states that the agent should get the key and then go to the door, and also avoid nails. We now demonstrate how we track the NFA state set for the top NFA. At the beginning of an episode, the agent is in the initial state of the MDP and the NFA is in state u_0. Thus, the NFA state set is initialized to $\{u_0\}$. This NFA state set will remain constant until the agent reaches an MDP state s' for which $\texttt{at(key)} \in L(s')$ (*i.e.*, the agent gets the key). Since the transition to u_1 is now possible in the NFA, but it is also possible to remain in u_0 because $\texttt{true}$ holds in s', the NFA state set is updated to $\{u_0, u_1\}$. The state set will then remain constant until the agent reaches the door.

We note that the above procedure may lead to an empty NFA state set on some NFAs and state sequences. For example, notice that in the bottom NFA in Figure 1b, there is no transition to follow when the agent enters a state with a nail. This occurs because once the agent is at a nail, the episode can never satisfy the advice formula regardless of how the rest of the episode proceeds. We call such situations *NFA dead-ends*. Since the advice may still be useful even if it has been violated, we handle NFA dead-ends as follows: if an NFA state set becomes empty, we revert it to the previous set. The NFA in Figure 1b will therefore continue to suggest that the agent avoid nails even if the agent already failed to do so.

Background Knowledge Functions

Advice like "get the key" would not be very useful if the agent had no notion of what behaviour might lead to the key. To give advice, we must presuppose that the advised agent has some capacity to understand and apply the advice. To this end, in this paper we assume that the agent has a *background knowledge function* $h_B : S \times A \times \mathsf{lit}(\Sigma) \to \mathbb{N}$, where $h_B(s, a, \ell)$ is an estimate of the number of primitive actions needed to reach the first state s' where the literal ℓ is true (*i.e.*, where $\ell \in L(s')$) if we execute a in s (note that a itself is not counted). Intuitively, h_B represents the agent's prior knowledge — which may not be perfectly accurate — about how to make ground atomic formulae either true or false.

Since h_B only provides estimates with respect to individual ground literals, we believe that for many applications it should be relatively easy to manually define (or learn, *e.g.* [7]) a reasonable h_B. For example, in the grid world environment, we define the background knowledge function as $h_B(s, a, \mathtt{at}(c)) = |\mathtt{pos}(agent, s).x - \mathtt{pos}(c, s).x| + |\mathtt{pos}(agent, s).y - \mathtt{pos}(c, s).y| + \Delta$ where $\mathtt{pos}(c, s)$ provides the coordinates of c in state s and Δ is equal to -1 if the action a points toward c from the agent's position and 1 if it does not. Hence, if the agent is three locations to the left of the key, $h_B(s, \text{right}, \mathtt{at}(\text{key}))$ will return 2 (right is the action to move right), even if there is a wall in the way. Furthermore, we defined $h_B(s, a, \neg\mathtt{at}(c))$ to be equal to 1 if $h_B(s, a, \mathtt{at}(c)) = 0$ and 0 otherwise.

Given any background knowledge function h_B, we can construct a function $h : S \times A \times \mathcal{L}_\Sigma \to \mathbb{N}$ that extends h_B to provide an estimate of the number of primitive actions needed to satisfy an arbitrary formula $\varphi \in \mathcal{L}_\Sigma$. We recursively define $h(s, a, \varphi)$ as follows: $h(s, a, \ell) = h_B(s, a, \ell)$ for $\ell \in \mathsf{lit}(\Sigma)$, $h(s, a, \psi \wedge \chi) = \max\{h(s, a, \psi), h(s, a, \chi)\}$, and $h(s, a, \psi \vee \chi) = \min\{h(s, a, \psi), h(s, a, \chi)\}$. In the next subsection, we describe how to drive the agent's exploration using h.

Advice-Based Action Selection in R-MAX

Model-based Reinforcement Learning solves MDPs by learning the transition probabilities and rewards. The R-MAX family of algorithms [2] are model-based methods that explore the environment by assuming that unknown transitions give maximal reward R_{max}. In practice, if R_{max} is big enough (and $\gamma < 1$), the agent ends up planning towards reaching the closest unknown transition. Instead, we propose to use the advice to plan towards *promising* unknown transitions.

Suppose the agent is at state s in the MDP, and we have kept track of the state sets that each NFA might be in. Intuitively, following the advice in s involves having the agent take actions that will move the agent through the edges of each NFA towards its accepting states. To this end, we begin by identifying *useful* NFA edges which may actually lead to progress towards the accepting states. For some NFA state u in an NFA state set, a transition from u to u' is useful if u is not an accepting state, and there exists a path in the NFA from u' to an accepting state that does not pass through u.

We now construct the *advice guidance* formula $\hat{\varphi}$ and the *advice warning* formula $\hat{\varphi}_w$. The advice guidance formula is given by the disjunction of all the formulae labelling useful edges in the NFAs. This means that $\hat{\varphi}$ will be satisfied by a transition in the MDP that achieves one of the formulae needed to transition over a useful edge. We define $\hat{h}(s, a) := h(s, a, \hat{\varphi})$ which can be used to rank actions based on an estimate of how close they are to making progress in satisfying the advice guidance formula $\hat{\varphi}$. While we omit the details, the advice warning formula $\hat{\varphi}_w$ is constructed similarly, but in such a way that it is satisfied by a transition in the MDP if and only if at least one transition is made in all of the NFAs. We then define the set $W(s) = \{a \in A(s) : h(s, a, \hat{\varphi}_w) \neq 0\}$. The idea is that W contains those actions which the evaluation function predicts will lead to an NFA dead-end.

We implemented a simple variant of an R-MAX algorithm that can take advantage of the advice. Instead of planning towards *the closest* unknown transition, we plan towards the closest unknown transition that is not in W and has minimum $\hat{h}$-value. If every unknown transition is in W, then we just go to the closest unknown transition with minimum $\hat{h}$-value.

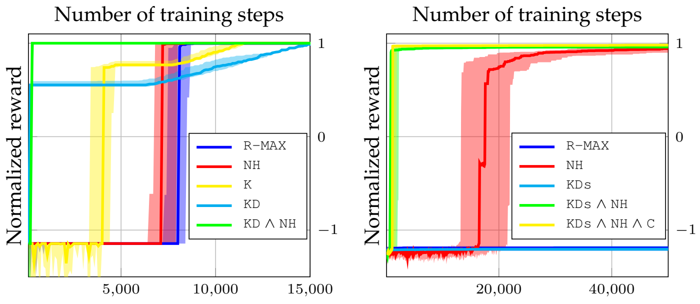
Convergence: As our approach plans towards unknown transitions (regardless of the advice), eventually all reachable transitions become known. By that point, the agent converges to a policy using its estimation of the transition probabilities. This policy is optimal with respect to $\mathcal{M}$ if the transitions are deterministic and near optimal otherwise [2].

4 Evaluation and Discussion

We tested our approach using various pieces of advice on grid world problems of the sort defined in Section 2, using the signature and background knowledge functions described in Sections 2 and 3. In addition to the domain elements of walls, keys, and nails described earlier, some problems also had *cookies* and *holes*. When the agent reaches a location with a cookie, it gets a reward of +10 and the cookie disappears. For reaching a hole, the reward is -1000 and the episode

Abbreviation	Advice formula (and informal meaning)
`R-MAX`	*No advice* Standard R-MAX.
`C`	$\forall(c \in \texttt{cookies}).\Diamond\texttt{at}(c)$ Get all the cookies.
`NH`	$\Box(\forall(x \in \texttt{nails} \cup \texttt{holes}).\neg\texttt{at}(x))$ Avoid nails and holes.
`K`	$\Diamond\texttt{at}(\texttt{key})$ Get the key.
`KD`	$\Diamond(\texttt{at}(\texttt{key}) \wedge \bigcirc\Diamond(\texttt{at}(\texttt{door})))$ Get the key and then go to the door.
`KDs`	$\forall(k \in \texttt{keys}).\Diamond(\texttt{at}(k) \wedge \bigcirc\Diamond(\exists(d \in \texttt{doors}).\texttt{at}(d)))$ For every key in the map, get it and then go to a door.

(a) Formula abbreviations for legends



(b) Nail room (25 × 50 version)

(c) Random grid world maps

Figure 2: In (b) and (c), reward is normalized so that 1 represents the best policy found on each map. The shaded areas represent the first and third quartiles. The formulae in the legends are explained by (a). Graphs are best viewed in colour.

ends. As a baseline, we reported the performance of standard R-MAX, which does not use any form of advice. To measure performance, we evaluated the agent's policy every 100 training steps. At training time, the agent explored the environment to learn a good model. At test time, we evaluated the best policy that the agent could compute using its current model (ignoring the unknown transitions). We used a discount factor of 1 in all experiments.

Figure 2b shows the median performance (over 20 independent trials) in a 25×50 version of the motivating example in Figure 1a. R-MAX, without advice, has poor initial performance, but converges to an optimal policy more quickly than when using the more detailed advice of `K` or `KD` (see Figure 2a for what these formulae are). However, `K` and `KD` quickly find a policy that follows the advice and then slowly converges to the optimal policy. As such, there is a trade-off between quickly learning the model (exploring nearby areas) and moving towards promising states suggested by advice. We also had an experiment with misleading advice (on a different map), where the agent was advised to go to a location far off any optimal path. The results were comparable to those seen for Figure 2b and are excluded due to lack of space.

We also randomly generated 10 grid world problems (Figure 1c shows one of them), each consisting of a sequence of four 25×25 rooms with doors between consecutive rooms. The agent always starts in the upper-left corner of the leftmost room and the red (*i.e.* goal) door is on the rightmost wall. Each space within a room had a $1/9$ probability of being each of a nail, hole, or wall. A key and three cookies were randomly placed in each room among the remaining empty spaces, such that each key, door, and cookie was reachable from the starting location. These grid world maps are challenging because there is sparse reward and it is fairly easy to die. Figure 2c shows the performance over the 10 maps using five different pieces of advice. We ran 5 trials per piece of advice on each map; the graph reports the median performance across both trials and maps. Without advice, the agent was never able to reach the last door in $50K$ training steps. Providing advice that the agent should get keys and go to doors (`KDs`) was also not enough, because the agent always fell in a hole before reaching its target. Stronger results were seen with safety-like advice to avoid holes and nails (`NH`), and the best performance was seen when this safety-like advice was combined with advice that guides agent progress ($\texttt{KDs} \wedge \texttt{NH}$ and $\texttt{KDs} \wedge \texttt{NH} \wedge \texttt{C}$). It is worth mentioning that none of these pieces of advice solves the problem for the agent, as the agent must still figure out how to turn the given advice into primitive actions, and handle cases where the advice is inappropriate (*e.g.* when the advice guides the agent towards cookies that are in a different room).

Future work: We plan to extend our work by prioritizing some parts of advice over others (for example, getting the key may be more important than getting every cookie), combining advice with hard constraints, experimenting with non-deterministic MDPs, and learning the background knowledge function h_B given a signature Σ. Finally, while the investigation here was limited to discrete MDPs, we also believe that many of the basic principles can be extended to a hybrid setting where discrete advice specified in LTL is used to guide search in a continuous space.

References

[1] J. Baier and S. McIlraith. Planning with first-order temporally extended goals using heuristic search. In *AAAI*, 2006.

[2] R. Brafman and M. Tennenholtz. R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 2002.

[3] G. De Giacomo, R. De Masellis, and M. Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *AAAI*, 2014.

[4] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 2009.

[5] S. Krening, B. Harrison, K. Feigh, C. Isbell, M. Riedl, and A. Thomaz. Learning from explanations using sentiment and advice in RL. *IEEE Transactions on Cognitive and Developmental Systems*, 2016.

[6] R. Maclin and J. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 1996.

[7] B. Peng, J. MacGlashan, R. Loftin, M. Littman, D. Roberts, and M. Taylor. A need for speed: Adapting agent action speed to improve task learning from non-expert humans. In *AAMAS*, 2016.

[8] A. Pnueli. The temporal logic of programs. In *FOCS*, 1977.

Bridging Computational Neuroscience and Machine Learning on Non-Stationary Multi-Armed Bandits

George Velentzas
School of Electerical and Computer Engineering
National Technical University of Athens
Athens, Greece
`geovelentzas@gmail.com`

Costas Tzafestas
School of Electerical and Computer Engineering
National Technical University of Athens
Athens, Greece
`ktzaf@cs.ntua.gr`

Mehdi Khamassi
Institute of Intelligent Systems and Robotics
Sorbonne Universités, UPMC Univ Paris 06, CNRS
F-75005 Paris, France
`mehdi.khamassi@upmc.fr`

Abstract

Fast adaptation to changes in the environment requires both natural and artificial agents to be able to dynamically tune an exploration-exploitation trade-off during learning. This trade-off usually determines a fixed proportion of exploitative choices (i.e. choice of the action that subjectively appears as best at a given moment) relative to exploratory choices (i.e. testing other actions that now appear worst but may turn out promising later). The problem of finding an efficient exploration-exploitation trade-off has been well studied both in the Machine Learning and Computational Neuroscience fields. Rather than using a fixed proportion, non-stationary multi-armed bandit methods in the former have proven that principles such as exploring actions that have not been tested for a long time can lead to performance closer to optimal – bounded regret. In parallel, researches in the latter have investigated solutions such as progressively increasing exploitation in response to improvements of performance, transiently increasing exploration in response to drops in average performance, or attributing exploration bonuses specifically to actions associated with high uncertainty in order to gain information when performing these actions. In this work, we first try to bridge some of these different methods from the two research fields by rewriting their decision process with a common formalism. We then show numerical simulations of a hybrid algorithm combining bio-inspired meta-learning, kalman filter and exploration bonuses compared to several state-of-the-art alternatives on a set of non-stationary stochastic multi-armed bandit tasks. While we find that different methods are appropriate in different scenarios, the hybrid algorithm displays a good combination of advantages from different methods and outperforms these methods in the studied scenarios.

Keywords: Bandits; Decision-making; Meta-learning; Active exploration; Kalman filter; Reinforcement learning; Multi-armed bandit

Acknowledgements

This research work has been partially supported by the EU-funded Project BabyRobot (H2020-ICT-24-2015, grant agreement no. 687831), by the Agence Nationale de la Recherche (ANR-12-CORD-0030 Roboergosum Project and ANR-11-IDEX-0004-02 Sorbonne-Universités SU-15-R-PERSU-14 Robot Parallearning Project), and by Labex SMART (ANR-11-LABX-65 Online Budgeted Learning Project).

1 Introduction

Selecting an action from a number of distinctive alternatives in unknown environments, where each action may return a reward in a stochastic manner, can be a challenging problem when the environment is non-stationary (i.e the rewards are changing in time). The adaptive learning strategy of the decision maker, especially when facing abrupt changes, requires a sophisticated method for the well known exploration-exploitation trade-off. Modeling the problem for implementing intelligent decision-makers is a subject mainly studied from the machine learning field, while the computational neuroscience field is concerned about finding models that fit human's or animal's behavior efficiently. In the former the task is to find a decision-making algorithm whose actions achieve optimal cumulative regret - the regret being defined as the difference between optimal performance and the accumulated reward. Solutions to the stationary problem have been studied thoroughly for more than a decade, but the non-stationary approach was a more recent subject of research [1,2], while its recent recapturing of attention is particularly due to its characteristic as a keystone in reinforcement learning. In parallel, computational neuroscience researches have also progressed in combining methods from reinforcement learning with exploration bonuses [3] and sometimes change-point detectors [4] to tackle the problem of fitting a human's or an animal's action decisions on bandit tasks with altering rewards in time.

With the dual goal of conceiving adaptive machines with cognitive components (e.g for human-robot interaction environments) and of generating new experimental predictions for neuroscience, we explored the impact of integrating approaches from both fields at the lower level of analysis for a simple artificial decision-maker. We tested the performance of several alternative algorithms on a set of different non-stationary bandit setups where we variated the most crucial components of a stochastic and changing environment. Here we present empirical evidence that our hybrid approach outperforms some state-of-the-art alternatives, rising questions about how the integration of neuroscience-based models and machine learning-models could have a positive impact on performance.

2 Problem Formulation and Algorithms

Stochastic multi armed-bandits can be considered as having a set of arms $\mathcal{K} = \{1, 2, ..., K\}$ of a casino machine, where a decision-maker chooses an action $a \in \mathcal{K}$ at every timestep $t \in \mathcal{T} = \{1, 2, ..., T\}$ of its lifespan T (also called the time horizon). He then receives a reward $r_t(a)$ with probability $p_t(a)$ and zero otherwise. While interested in arm's expected value of reward, without loss of generality, we can assume Bernoulli arms where $\forall(a, t) \in \mathcal{K} \times \mathcal{T}$ the rewards are binary, $r_t(a) \in \{0, 1\}$. By choosing Bernoulli arms, the expected value $\mathbb{E}[r_t(a)]$ for every arm reward is equal to the probability $p_t(a)$. When the environment is stationary, $p_{t+1}(a) = p_t(a)$ for all time steps. For non-stationary environments the above rule does not stand. Specifically in drifting environments $|p_{t+1}(a) - p_t(a)| < \epsilon$, where ϵ is a small constant, while in abruptly changing environments $\exists t : |p_{t+1}(a) - p_t(a)| > \delta$, where δ is a sufficiently large value in terms of probabilities. Both ϵ and δ can be used as a measure of the environment dynamics. Denoting a^* the optimal arm to choose, the regret at every time step is then defined as $\mathbb{E}[r_t(a^*) - r_t(a)]$.

A first important note is that in most bandit methods, from either machine learning or computational neuroscience, the action a_t to be taken at the time episode t can be chosen after comparing between all actions a combination of an empirical measure of each action's value $\mu_t(a)$ and some measure of uncertainty for this value $\sigma_t(a)$, according to the following general equation:

$$a_t = \mathcal{F}(\beta_t(\mu_t(a) + \phi\sigma_t(a))) \tag{1}$$

In the classical family of bandit methods initially developed in machine learning - called *UCB* for Upper Confidence Bound [5] -, function $\mathcal{F}$ is an argmax, parameters β and ϕ are fixed to 1, and $\mu_t(a)$, $\sigma_t(a)$ are empirically estimating the average reward yielded by action a and the uncertainty associated to this estimate. Then the method differ in the way these estimates are updated, using for instance a *Sliding Window* (SW-UCB) or *Discounted* estimates (D-UCB) [1]. In a more recent work, the *Kalman Filter Multi-Armed Normal Bandit* (KF-MANB) proposed in [2] is also another empirically optimal algorithm using an observational noise parameter σ_{ob} and a transitional noise parameter σ_{tr} to estimate $\mu_t(a)$ and $\sigma_t(a)$ at each timestep, which are then used as the parameters of a normal distribution $\mathcal{N}(\mu_t(a), \sigma_t(a))$ for each arm. Then a sample s_a from each distribution is taken, and the action choices are made according the rule $a_t = \arg\max_a (s_a)$.

In computational neuroscience models for bandit tasks [3, 6], function $\mathcal{F}$ is a Boltzmann softmax followed by sampling from the resulting action probability distribution, parameters β and ϕ are either fixed or estimated from each human subject's behavior during the task, $\mu_t(a)$ is the *Q-value* of action a learned through reinforcement learning - most often with a discounted factor $\gamma = 0$ because the task is not sequential - and $\sigma_t(a)$ is an exploration bonus. In some models, $\mu_t(a)$ can be transiently adapted faster through the use of change point detectors which identify abrupt task shifts that produce a higher noise than the expected observational noise [4]. Interestingly, this has also been recently applied to bandit methods in Machine Learning (*e.g.* the *Adapt-Eve* algorithm [7]). Finally, some models propose to dynamically tune through *meta-learning* the algorithm's parameters as a function of variations of performance [8, 9]: for instance, increasing the inverse temperature β_t in the softmax function to promote exploitation when the average performance improves, and decreasing it to promote exploration when performance drops indicate abrupt changes in the task.

In previous work, we have applied this meta-learning principle in an algorithm here referred to as *MLB* to dynamically tune β_t in a simple multi-armed bandit scenario involving the interaction between a simulated human and a robot [10]. Formally, function $\mathcal{F}$ is a Boltzmann softmax with parameter $\phi = 0$ (*i.e.* no uncertainty term is used as exploration bonus here). The immediate reward r_t received from action $a = a_t$ is then used to update the estimated action's expected value $\mu_t(a_t)$ of the action taken, using the update rule of a low-pass filter, as $Q_{t+1}(a) = (1 - \alpha_Q)Q_t(a) + \alpha_Q r_t$. Furthermore, following [8], a similar update rule applies to estimate mid-term and long-term reward running averages according to:

$$\overline{r}_t = \alpha_m r_t + (1 - \alpha_m)\overline{r}_{t-1} \text{ and } \overline{\overline{r}}_t = \alpha_\ell \overline{r}_t + (1 - \alpha_\ell)\overline{\overline{r}}_{t-1} \tag{2}$$

where α_m and α_ℓ are two different learning rates. β_t is then updated as

$$\beta_{t+1} = max\{\beta_t + \eta(\overline{r}_t - \overline{\overline{r}}_t)), 0\} \tag{3}$$

With the above update rules, β_t increases when the mid-term reward is greater than the long-term. One can view this as an evidence that the recent actions are more optimal, therefore exploitation may be increased. In the opposite case when $\overline{r}_t < \overline{\overline{r}}_t$, the recent actions denote that the present policy has lower performance in getting rewards compared to the past, therefore exploration should be encouraged by reducing β_t.

3 Hybrid Meta-learning with Kalman Filters - MLB-KF

After showing a common way to formalize bandit methods in machine learning and computational neuroscience, the objective of the present paper is to illustrate how a hybrid method combining ideas from the two fields that we have recently proposed in [11] can yield a promising performance in a variety of non-stationary multi-armed bandit tasks. The hybrid model integrates the sibling kalman filters of KF-MANB and the core of MLB. To do so, action values learned through reinforcement learning in *MLB* are here simply replaced within the softmax function with a linear combination (with $\phi > 0$) of the mean $\mu_t(a)$ and the standard deviation $\sigma_t(a)$ of each arm's distribution according to the sibling kalman filters of KF-MANB. The algorithm also uses two constants σ^2_{ob} and σ^2_{tr}, which relate to the stochastic part and the non-stationary part of the environment respectively. The values for means $\mu_1(a)$, variances $\sigma^2_1(a)$, and β_t are initialized. Following [2], but writing equations down on a different way, the reward $r_t(a)$, for $a = a_t$, is then used to update the parameters of the respective arm with

$$\mu_{t+1}(a) = v_t(a)r_t(a) + (1 - v_t(a))\mu_t(a) \text{ and } \sigma^2_{t+1}(a) = v_t(a)\sigma^2_{ob} \tag{4}$$

while for arms $a \neq a_t$ the means $\mu_{t+1}(a)$ maintain their previous value, and a transitional variance is added to $\sigma^2_{t+1}(a)$ in order to enhance exploration of non-chosen arms as shown below

$$\mu_{t+1}(a) = \mu_t(a) \text{ and } \sigma^2_{t+1}(a) = \sigma^2_t(a) + \sigma^2_{tr} \tag{5}$$

In the above equations $v_t(a)$ are dynamically tuned learning rates for each arm, which may mirror some adaptation procedures of learning rate studied in a computational neuroscience bandit task (*e.g.* [12]). Here, for KF-MANB, it follows:

$$v_t(a) = (\sigma_t(a)^2 + \sigma^2_{tr})/(\sigma_t(a)^2 + \sigma^2_{tr} + \sigma^2_{ob}) \tag{6}$$

The mid- and long-term rewards are also calculated like in [8] and used to update the inverse temperature parameter β_t according to the update rule of MLB [10]. When the uncertainty of an arm's action value increases, the respective arm should be explored more. However when the environment suggests that no evidence of change is observed, KF-MANB suffers from exploration since a transitional variance is always added to the non-pulled arms. This can also be seen in Fig.1 (right) of simulations section. MLB is also expected to display large regret at cases where a non-optimal arm becomes optimal after long periods of stationarity, since the "rising" arm may never be chosen. The above implementation is a trade-off between these two cases, as we empirically present evidence that it incorporates both the benefits of MLB's fast adaptation and small regret on many test cases, as also the robustness and good performance of KF-MANB.

Algorithm 1 Hybrid MLB-KF

```
Choose parameters $\alpha_m$, $\alpha_\ell$, $\eta$, $\phi$, $\sigma_{ob}$, $\sigma_{tr}$
Initialize $\beta_1$, and $\mu_1(a)$, $\sigma_1(a)$ $\forall a \in \mathcal{K}$
for $t = 1, 2, ..., T$ do
    select action $a_t \in \mathcal{K}$ from distribution of (1) with softmax
    observe the immediate reward $r_t$ and update $\overline{r}_t$, $\overline{\overline{r}}_t$ with (2)
    for $a = a_t$ update $\mu_{t+1}(a)$, $\sigma^2_{t+1}(a)$ with (4),(6)
    for all arms $a \neq a_t$ update $\mu_{t+1}(a)$, $\sigma^2_{t+1}(a)$ with (5)
    update $\beta_{t+1}$ with (3)
end for
```

4 Numerical Simulations

Testing an algorithm on a scenario that was also used in order to tune its parameters, should restrict its evaluation of performance to concern only this specific environment. Here we avoided this bias by testing the algorithms on many different scenarios, altering the most crucial components of a stochastic and non-stationary multi armed bandit, while keeping the parameters that were tuned on a simple mid-case. For the mid-case we considered 5 arms, with initial arm probabilities of 0.8, 0.4, 0.3, 0.2, 0.1 for each arm respectively, with a circular shift of those at every 2000 time steps, in a total of 10000. After optimization, the parameters chosen for D-UCB were $\xi = 0.22$ and $\gamma = 0.9999$, for SW-UCB $\xi = 0.3$ and $\tau = 998$, for Adapt-EvE $\delta = 0.13$, $\lambda = 40$, $t_m = 50$ and used UCBT as decision-makers and as meta-bandit. For KF-MANB $\sigma_{ob} = 0.2$, $\sigma_{tr} = 0.01$, initializing all means and variances to 0.5. For MLB $\alpha_Q = 0.14$, $\alpha_m = 1/15$, $\alpha_\ell = 1/350$, $\eta = 0.44$. For MLB-KF we kept the parameters found for MLB and KF-MANB and only tuned $\phi = 1.5$.

For problem set I, we considered that a global change point cp, may occur at every time step t with a constant probability h (i.e $p_t(cp) = h$). When a change point occurs, all arm probabilities are then re-sampled from a uniform distribution in [0, 1]. For each subproblem (using only the set of probabilities generated for each arm with the above type of random walk), we run each algorithm for 20 sub-sessions, then regenerated another problem set with the same probability h, and repeated the procedure for 100 hyper-sessions. We then calculated the average total cumulative regret with the assumption that the results are representative due to central limit theorem. We increased h and repeated all the above, for $h \in [2/10000, 8/10000]$ with a step of $1/10000$. Therefore, an average of 2 to 8 global change points occur (we have an horizon of 10000) for every step. As seen in Fig 1 (left), MLB exhibited the best performance for small h, KF-MANB demonstrated both robustness and low regret in all cases and MLB-KF inherited the best of these performances by following the regret dynamics of the most optimal algorithm between the two. In problem set II we explored cases where the difference Δ_g between the expected reward of the optimal action and the second best action varied. The best optimal arm always had a probability of 0.8, the second best $0.8-\Delta_g$, the third $0.8-2\Delta_g$ and so forth. For our experiments we begun with $\#cp = 1$ (where $\#cp$ denotes the number of change points). This change point occurred at $t = T/2$, when the probability of the best arm dropped from 0.8 to 0.8-Δ_g, the probability of the second best dropped from 0.8-Δ_g to 0.8-2Δ_g (and so forth for other intermediate arms), while the probability of the worst arm increased from 0.8-4Δ_g to 0.8. We altered Δ_g from 0.02 to 0.2 with a step size of 0.02 for each independent test case, while we simulated all algorithms on each case for 200 sessions, observing the average final cumulative regret. We then increased $\#cp$ by one, fixed the change point time steps to be evenly distributed over the timesteps, and repeated the procedure. In Fig.1 (middle) a visualization of the performance for each algorithm is presented, for different gaps Δ_g (rows), and $\#cp$ (columns). The results once again provided evidence that MLB-KF combines the behavior of both MLB and KF-MANB in a satisfactory manner, denoting the best performance on average. Finally, cases where the best optimal action becomes the less optimal, cases where the less optimal action becomes the optimal as also sequential alteration of those were tested in problem set III. MLB adapted greatly in Best-to-Worst scenarios and D-UCB in Worst-to-Best. However the overall average performance of MLB-KF was the empirically optimal as described in [11] (not shown here).

In order to highlight how parameters $v_t(a)$ change in time, a simple two-armed bandit case using KF-MANB, where $p_t(a_1)$ was changing from 0.75 to 0.25 every 50 timesteps and $p_t(a_2) = 1 - p_t(a_1)$, is also shown in Fig. 1 (right) for an average of 5 sessions. For the interval between timesteps 50-100, meta-parameter $v_t(a_1)$ increases since the most chosen arm was a_2. Even though the estimated reward rate of arm a_1 in this interval differs from the true rate, a more precise estimation is not needed in order to make the most optimal decision. However, evidence about the arm's reward rate in this interval is not provided, therefore $v_t(a_1)$ increases, and then decreases as the arm is once again frequently chosen.

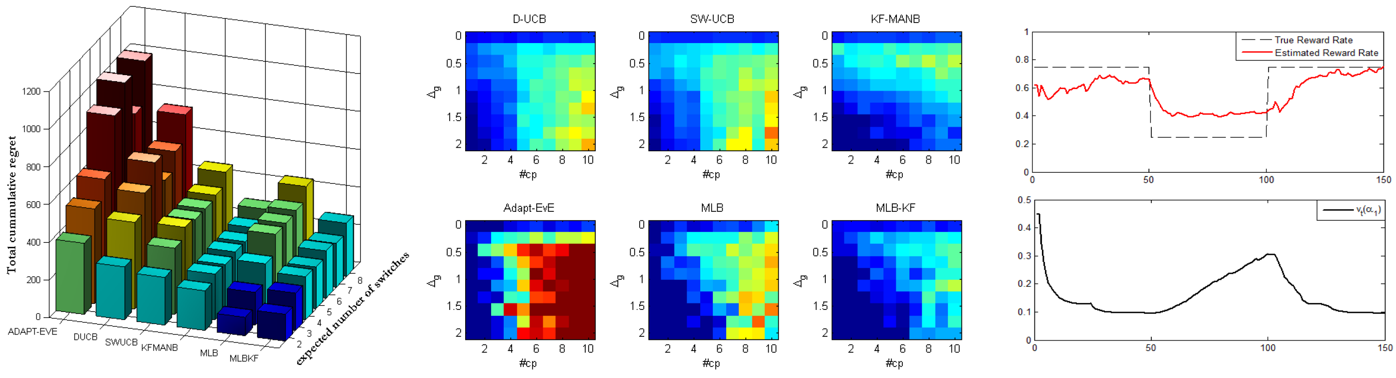


Figure 1: left: total regret for different values of expected number of change points h, middle: total cumulative regret for different gaps Δ_g and number of switches $\#cp$ (thresholded for low and high values), right: estimated action value $\mu(a_1)$ vs true value (top) and parameter $v_t(a_1)$ of KF-MANB for a two-armed bandit case (bottom)

5 Conclusion and Discussion

In this work we presented empirical evidence that a hybrid algorithm which makes use of a classical computational neuroscience approach to properly tune exploration with meta-learning, combined with sibling Kalman filters proposed in Machine Learning to estimate each arm's action value and add a measure of uncertainty as an exploration bonus, can lead to an adaptive learning strategy which efficiently manages the exploration-exploitation trade-off dilemma. However we still need to do the comparison more systematically for many different tasks (different numbers of arms, different stochasticity levels, different amplitudes between best and second-best arm, abrupt task shifts versus progressive drifts, different variation budget [13]). Nevertheless, such a bridge between the fields of computational neuroscience and machine learning seems promising.

Besides, each development of a new algorithm displaying efficient adaptation performance is an opportunity to sketch some new experimental predictions for neuroscience. Here, an interesting aspect of the hybrid algorithm is that it combines dynamic tuning of a global exploration rate β_t through meta-learning and adaptive tuning of action specific learning rates $v_t(a)$ based on uncertainty variations in a Kalman filter. The resulting two timeseries may be less colinear than if they had been generated by the same method, which can be useful to generate regressors for model-based analyses of neurocience data. In addition, the resulting learning rates $v_t(a)$ may follow a different dynamics - briefly illustrated in Fig. 1 (right) - than other computational methods of learning rate studied in neuroscience bandit tasks, such as adaptations as a function of reward volatility [12]. Again, this may constitute an interesting experimental prediction to explore and compare in the future with human behavioral and neural data.

Finally, it is interesting to note recent progresses in machine learning using meta-learning combined with deep reinforcement learning [14]. Their deep learning approach is promising to not only dynamically tune some parameters but learn at a more global level the dynamics/statistics of the task. In addition, the method produces generalization to different but statistically related bandit tasks. All these results also suggest that a common ground between meta-learning and reinforcement learning can help find general methods to not only solve single-step bandit tasks, but also generalize to multi-step reinforcement leaning problems.

References

[1] A. Garivier and E. Moulines, "On upper-confidence bound policies for non-stationary bandit problems," *Algorithmic Learning Theory*, pp. 174–188, 2011.

[2] O.-C. Granmo and S. Berg, *Solving Non-Stationary Bandit Problems by Random Sampling from Sibling Kalman Filters*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 199–208.

[3] N. D. Daw, J. P. O'Doherty, P. Dayan, B. Seymour, and R. J. Dolan, "Cortical substrates for exploratory decisions in humans," *Nature*, vol. 441, no. 7095, pp. 876–879, 2006.

[4] R. C. Wilson, M. R. Nassar, and J. I. Gold, "Bayesian online learning of the hazard rate in change-point problems," *Neural computation*, vol. 22, no. 9, pp. 2452–2476, 2010.

[5] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[6] M. J. Frank, B. B. Doll, J. Oas-Terpstra, and F. Moreno, "Prefrontal and striatal dopaminergic genes predict individual differences in exploration and exploitation," *Nature neuroscience*, vol. 12, no. 8, pp. 1062–1068, 2009.

[7] C. Hartland, S. Gelly, N. Baskiotis, O. Teytaud, and M. Sebag, "Multi-armed bandit, dynamic environments and meta-bandits," in *NIPS 2006 Workshop on Online trading between exploration and exploitation*, Whistler, Canada, 2006.

[8] N. Schweighofer and K. Doya, "Meta-learning in reinforcement learning." *Neural Networks*, vol. 16, no. 1, pp. 5–9, 2003.

[9] M. Khamassi, P. Enel, P. Dominey, and E. Procyk, "Medial prefrontal cortex and the adaptive regulation of reinforcement learning parameters," *Progress in Brain Research*, vol. 202, pp. 441–464, 2013.

[10] M. Khamassi, G. Velentzas, T. Tsitsimis, and C. Tzafestas, "Active exploration and parameterized reinforcement learning applied to a simulated human-robot interaction task," in *IEEE Robotic Computing 2017*, Taipei, Taiwan, 2017.

[11] G. Velentzas, C. Tzafestas, and M. Khamassi, "Bio-inspired meta-learning for active exploration during non-stationary multi-armed bandit tasks," in *IEEE Intelligent Systems Conference 2017*, London, UK, 2017.

[12] T. E. Behrens, M. W. Woolrich, M. E. Walton, and M. F. Rushworth, "Learning the value of information in an uncertain world," *Nature neuroscience*, vol. 10, no. 9, pp. 1214–1221, 2007.

[13] O. Besbes, Y. Gur, and A. Zeevi, "Optimal exploration-exploitation in a multi-armed-bandit problem with non-stationary rewards," *arXiv preprint arXiv:1405.3316*, 2014.

[14] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv preprint arXiv:1611.05763*, 2016.

Prediction under Uncertainty in Sparse Spectrum Gaussian Processes with Applications to Filtering and Control

Yunpeng Pan
School of Aerospace Engineering
Georgia Institute of Technology
`ypan37@gatech.edu`

Xinyan Yan
School of Interactive Computing
Georgia Institute of Technology
`xyan43@gatech.edu`

Evangelos A. Theodorou
School of Aerospace Engineering
Georgia Institute of Technology
`evangelos.theodorou@gatech.edu`

Byron Boots
School of Interactive Computing
Georgia Institute of Technology
`bboots@cc.gatech.edu`

Abstract

In many sequential prediction and decision-making problems such as Bayesian filtering and probabilistic model-based planning and control, we need to cope with the challenge of *prediction under uncertainty*, where the goal is to compute the predictive distribution $p(y)$ given an input distribution $p(x)$ and a probabilistic model $p(y|x)$. Computing the exact predictive distribution is generally intractable. In this work, we consider a special class of problems in which the input distribution $p(x)$ is a multivariate Gaussian, and the probabilistic model $p(y|x)$ is learned from data and specified by a sparse spectral representation of Gaussian processes (SSGPs).

SSGPs are a powerful tool for scaling Gaussian processes (GPs) to large datasets by approximating the covariance function using finite-dimensional random Fourier features. Existing SSGP algorithms for regression assume deterministic inputs, precluding their use in many sequential prediction and decision-making applications where accounting for input uncertainty is crucial. To address this *prediction under uncertainty* problem, we propose an exact moment-matching approach with closed-form expressions for predictive distributions. Our method is more general and scalable than its standard GP counterpart, and is naturally applicable to multi-step prediction or uncertainty propagation. We show that our method can be used to develop new algorithms for Bayesian filtering and stochastic model predictive control, and we evaluate the applicability of our method with both simulated and real-world experiments.

Keywords: Sequential prediction, Gaussian processes, planning and control, Bayesian filtering, probabilistic model-based reinforcement learning, approximate inference

1 Introduction

The problem of *prediction under uncertainty*, appears in many fields of science and engineering that involve sequential prediction and decision-making including state estimation, time series prediction, stochastic process approximation, and planning and control. In these problems, uncertainty can be found in both the predictive models and the model's inputs. Formally, we are often interested in finding the probability density of a prediction y, given a distribution $p(x)$ and a probabilistic model $p(y|x)$. By marginalization,

$$p(y) = \int p(y|x)p(x)\,\mathrm{d}x. \tag{1}$$

Unfortunately, computing this integral exactly is often intractable. In this paper, we tackle a subfamily of (1) where: 1) the probabilistic model is learned from data and specified by a sparse spectrum representation of a Gaussian process (SSGP); and 2) the input x is normally distributed.

1.1 Related work

Gaussian Process (GP) inference with uncertain inputs has been addressed by [2, 8], and extended to the multivariate outputs by [10]. These methods have led to the development of many algorithms in reinforcement learning [15, 4], Bayesian filtering [9, 6], and smoothing [5]. However, these approaches have two major limitations: 1) they are not directly applicable to large datasets, due to the polynomial (in data samples) time complexity for exact inference [16]; and 2) analytic moment expressions, when used, are restricted to squared exponential (SE) or polynomial kernels [10] and cannot be generalized to other kernels in a straightforward way.

A common method for approximating large-scale kernel machines is through random Fourier features [14]. The key idea is to map the input to a low-dimensional feature space yielding fast linear methods. In the context of GP regression (GPR), this idea leads to the sparse spectrum GPR (SSGPR) algorithm [11]. SSGP has been extended in a number of ways for, e.g. incremental model learning [7], and large-scale GPR [3]. However, to the best of our knowledge, prediction under uncertainty for SSGPs has not been explored. Although there are several alternative approximations to exact GP inference including approximating the posterior distribution using inducing points, comparing different GP approximations is not the focus of this paper.

1.2 Applications

We consider two problems that involve sequential prediction and decision-making: Bayesian filtering and stochastic model predictive control (MPC). The goal of Bayesian filtering is to infer a hidden system state from observations through the recursive application of Bayes' rule. GP-based assumed density filtering (ADF) with SE kernels has been developed by [6], which has demonstrated superior performance compared to other GP-based filters [9]. We extend this work with a highly efficient SSGP-based ADF approach.

The goal of stochastic MPC is to find finite horizon optimal control at each time instant. Due to the high computational cost of GP inference and real-time optimization requirements in MPC, most GP-based control methods [4, 13] are restricted to episodic reinforcement learning tasks. To cope with this challenge, we present an SSGP-based MPC approach that is fast enough to perform optimization and model adaptation on-the-fly.

1.3 Our contributions

- We propose an exact moment-matching approach to prediction under uncertainty in SSGPs with closed-form expressions for the predictive distribution. Compared to previous GP counterparts, our method: 1) is more scalable, and 2) can be generalized to any continuous shift-invariant kernels with a Fourier feature representation.
- We demonstrate successful applications of the proposed approach to 1) recursive Bayesian filtering and 2) stochastic model predictive control.

2 Sparse Spectral Representation of Gaussian Processes

Consider the task of learning the function $f : \mathbf{R}^d \to \mathbf{R}$, given data $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$, with each pair related by

$$y = f(x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2), \tag{2}$$

where ϵ is IID additive Gaussian noise. Gaussian process regression (GPR) is a principled way of performing Bayesian inference in function space, assuming that function f has a prior distribution $f \sim \mathcal{GP}(m, k)$, with mean function $m : \mathbf{R}^d \to \mathbf{R}$ and kernel $k : \mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}$. Without loss of generality, we assume $m(x) = 0$. Exact GPR is challenging for large datasets due to its $O(n^3)$ time and $O(n^2)$ space complexity [16], which is a direct consequence of having to store and invert an $n \times n$ Gram matrix.

Random features can be used to form an unbiased approximation of continuous shift-invariant kernel functions, and are proposed as a general mechanism to accelerate large-scale kernel machines [14], via explicitly mapping inputs to low-dimensional feature space. Based on Bochner's theorem, the Fourier transform of a continuous shift-invariant positive definite kernel $k(x, x')$ is a proper probability distribution $p(\omega)$ [14], which leads to an unbiased approximation of k: $k(x, x') \approx \frac{1}{m}\sum \phi_{\omega_i}(x)\phi_{\omega_i}(x')^*$, where random frequencies $\{\omega_i\}_{i=1}^m$ are drawn IID from $p(\omega)$. Considering ϕ_ω can be replaced by sinusoidal functions since both $p(\omega)$ and $k(x, x')$ are reals, and concatenating features $\{\phi_{\omega_i}\}_{i=1}^m$ into a succinct vector form, an approximation for $k(x, x')$ is expressed as:

$$k(x,x') \approx \phi(x)^T\phi(x'),\ \phi(x) = \begin{bmatrix}\phi^c(x)\\ \phi^s(x)\end{bmatrix},\phi_i^c(x) = \sigma_k \cos(\omega_i^T x),\ \phi_i^s(x) = \sigma_k \sin(\omega_i^T x),\ \omega_i \sim p(\omega), \tag{3}$$

where σ_k is a scaling coefficient. For the commonly used Squared Exponential (SE) kernel: $k(x,x') = \sigma_f^2 \exp(-\frac{1}{2}\|x-x'\|_{\Lambda^{-1}})$, $p(\omega) = \mathcal{N}(0,\Lambda^{-1})$ and $\sigma_k = \frac{\sigma_f}{\sqrt{m}}$, where the coefficient σ_f and the diagonal matrix Λ are the hyperparameters. Spectral densities $p(\omega)$ and scaling coefficients σ_k for other continuous shift-invariant kernels can be derived similarly. Based on the feature map, SSGP is defined as

Definition 1. *Sparse Spectrum GPs (SSGPs) are GPs with kernels defined on the finite-dimensional and randomized feature map ϕ* (3)*:*

$$\hat{k}(x,x') = \phi(x)^T\phi(x') + \sigma_n^2\delta(x-x'), \tag{4}$$

where the function δ is the Kronecker delta function, to account for the additive zero mean Gaussian noise in (2)*.*

Because of the explicit finite-dimensional feature map (3), each SSGP is equivalent to a Gaussian distribution over the weights of features $w \in \mathbf{R}^{2m}$. Assuming that prior distribution of weights w is $\mathcal{N}(0,\Sigma_w)$, and the feature map is fixed, after conditioning on the data $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$ the posterior distribution of w is

$$w \sim \mathcal{N}(\alpha,\ \sigma_n^2 A^{-1}),\quad \alpha = A^{-1}\Phi Y,\quad A = \Phi\Phi^T + \sigma_n^2\Sigma_\alpha^{-1}, \tag{5}$$

which can be derived through Bayesian linear regression. In (5), the column vector Y and the matrix Φ are specified by the data $\mathcal{D}$: $Y = [y_1 \quad \dots \quad y_n]^T$, $\Phi = [\phi(x_1) \quad \dots \quad \phi(x_n)]$. Consequently, the posterior distribution over the output y in (2) at a test point x is *exactly Gaussian*, in which the posterior variance explicitly captures the model uncertainty in prediction with input x:

$$p(y|x) = \mathcal{N}(\alpha^T\phi(x),\ \sigma_n^2 + \sigma_n^2\|\phi(x)\|_{A^{-1}}^2). \tag{6}$$

This SSGP regression method is proposed in [11]. Its time complexity is $O(nm^2 + m^3)$, which is significantly more efficient than standard GPR's $O(n^3)$ when $m \ll n$.

Remark: It's worth noting that the method proposed in this paper is not tied to specific algorithms for SSGP regression such as linear Bayesian regression [11], but accounts for any SSGP with specified feature weights distribution (5), where posterior α and A can be computed by any means. Variations on A include sparse approximations by a low rank plus diagonal matrix, or iterative solutions by optimization methods like doubly stochastic gradient descent [3].

3 Prediction under Uncertainty

We present an analytic moment-based approach to (1) under two conditions: 1) the uncertain input is normally distributed: $x \sim \mathcal{N}(\mu,\Sigma)$, and 2) probabilistic models are in the form of (6) specified by SSGPs. Despite these conditions, evaluating the integral in (1) is still intractable. In this work, we approximate the true predictive distribution $p(y)$ by a Gaussian via computation of the exact moments in closed-form. We consider multivariate outputs by utilizing conditionally independent scalar models for each output dimension, *i.e.*, assuming for outputs in different dimension y_a and y_b, $p(y_a, y_b|x) = p(y_a|x)p(y_b|x)$. For notational simplicity, we suppress the dependency of $\phi(x)$ on x, and treat y as a scalar by default.

In the following we present our method, SSGP-exact moment matching (SSGP-EMM). We derive 1) the predictive mean $\mathbf{E}\,y$; 2) the predictive variance $\mathbf{Var}\,y$ and covariance $\mathbf{Cov}(y_a, y_b)$, which in the multivariate case correspond to the diagonal and off-diagonal entries of the predictive covariance matrix; and 3) the cross-covariance between input and prediction $\mathbf{Cov}(x, y)$.

Using the expressions for SSGP (3), (6), and the law of total expectation, the predictive mean becomes

$$\mathbf{E}\,y = \mathbf{E}\,\mathbf{E}(y|x) = \mathbf{E}\left(\alpha^T\phi\right) = \alpha^T\,\mathbf{E}\begin{bmatrix}\phi^c\\ \phi^s\end{bmatrix},\quad \mathbf{E}\,\phi_i^c = \sigma_k\,\mathbf{E}\cos(\omega_i^T x),\quad \mathbf{E}\,\phi_i^s = \sigma_k\,\mathbf{E}\sin(\omega_i^T x), \tag{7}$$

where $i = 1,\dots,m$, and in the nested expectation $\mathbf{E}\,\mathbf{E}(y|x)$, the outer expectation is over the input distribution $p(x) = \mathcal{N}(\mu,\Sigma)$, and the inner expectation is over the conditional distribution $p(y|x)$ (6).

By observing (7), we see that the expectation of sinusoids under the Gaussian distribution is the key to computing the predictive mean. Thus we state the following proposition:

Proposition 1. *The expectation of sinusoids over multivariate Gaussian distributions: $x \sim \mathcal{N}(\mu,\Sigma)$, $x \in \mathbf{R}^d$, i.e., $p(x) = (2\pi)^{-\frac{d}{2}}(\det\Sigma)^{-\frac{1}{2}}\exp(-\frac{1}{2}\|x-\mu\|_{\Sigma^{-1}}^2)$, can be computed analytically:*

$$\mathbf{E}\cos(\omega^T x) = \exp(-\frac{1}{2}\|\omega\|_\Sigma^2)\cos(\omega^T\mu),\quad \mathbf{E}\sin(\omega^T x) = \exp(-\frac{1}{2}\|\omega\|_\Sigma^2)\sin(\omega^T\mu).$$

The proof is omitted. The predictive mean (7), variance, and covariance between different outputs are derived using Proposition 1. By the law of total variance, the predictive variance is

$$\mathbf{Var}\,y = \mathbf{E}\,\mathbf{Var}(y|x) + \mathbf{Var}\,\mathbf{E}(y|x)\alpha^T\Psi\alpha - (\mathbf{E}\,y)^2, \tag{8}$$

where Ψ is defined as the expectation of the outer product of feature vectors over input distribution $p(x)$. Specifically, we compute Ψ by applying the product-to-sum trigonometric identities:

$$\mathbf{E}\left(\phi\phi^T\right) = \Psi = \begin{bmatrix}\Psi^{cc} & \Psi^{cs} \\ \Psi^{sc} & \Psi^{ss}\end{bmatrix}, \qquad \Psi^{cc}_{ij} = \frac{\sigma_k^2}{2}\left(\mathbf{E}\left(\cos(\omega_i+\omega_j)^T x\right) + \mathbf{E}\left(\cos(\omega_i-\omega_j)^T x\right)\right),$$

$$\Psi^{ss}_{ij} = \frac{\sigma_k^2}{2}\left(\mathbf{E}\left(\cos(\omega_i-\omega_j)^T x\right) - \mathbf{E}\left(\cos(\omega_i+\omega_j)^T x\right)\right), \quad \Psi^{cs}_{ij} = \frac{\sigma_k^2}{2}\left(\mathbf{E}\left(\sin(\omega_i+\omega_j)^T x\right) - \mathbf{E}\left(\sin(\omega_i-\omega_j)^T x\right)\right) \tag{9}$$

where $\Psi^{cc}, \Psi^{ss}, \Psi^{cs}$ are $m \times m$ matrices, and $i, j = 1, \ldots, m$, on whose terms Proposition 1 can be directly applied.

Next we derive the covariance for different output dimensions for multivariate prediction. These correspond to the off-diagonal entries of the predictive covariance matrix. We show that, despite the conditional independence assumption for different outputs given a deterministic input, outputs become coupled with uncertain input. Using the law of total covariance, the covariance is:

$$\mathbf{Cov}(y_a, y_b) = \mathbf{Cov}\left(\mathbf{E}(y_a|x), \mathbf{E}(y_b|x)\right) = \mathbf{E}\left(\mathbf{E}(y_a|x), \mathbf{E}(y_b|x)\right) - (\mathbf{E}\, y_a)(\mathbf{E}\, y_b) = \alpha_a^T \Psi_{ab} \alpha_b - (\alpha_a^T \mathbf{E}\, \phi_a)(\alpha_b^T \mathbf{E}\, \phi_b) \tag{10}$$

where matrix Ψ_{ab} is the expectation of the outer product of feature vectors corresponding to different feature maps ϕ_a, ϕ_b for outputs y_a, y_b, computed similarly as in (3) with corresponding random frequencies $\{\omega_i\}$, and the scaling coefficient σ_k (3). α_a and α_b are the corresponding weight vectors for y_a and y_b (6). Compared to the expression for the variance of a single output in (8), the term $\mathbf{E}\left(\mathbf{Cov}(y_a|x)\, \mathbf{Cov}(y_b|x)\right)$ that is included in the law of total covariance is neglected due to the assumption of conditional independence of different outputs (§2), so (10) does not have the corresponding first two terms in (8).

Finally, we compute the covariance between input and each output dimension. Invoking the law of total covariance:

$$\mathbf{Cov}(x, y) = \mathbf{Cov}(x, \mathbf{E}(y|x)) = \mathbf{E}\left(x\, \mathbf{E}(y|x)\right) - (\mathbf{E}\, x)(\mathbf{E}\, y) = \Upsilon\alpha - (\mathbf{E}\, y)\mu, \tag{11}$$

where matrix Υ is the expectation of the outer product of the input x and the feature vector $\phi(x)$ over input distribution $x \sim \mathcal{N}(\mu, \Sigma)$:

$$\mathbf{E}(x\phi^T) = \Upsilon = \left[\Upsilon_1^c \quad \ldots \quad \Upsilon_m^c \quad \Upsilon_1^s \qquad \ldots \quad \Upsilon_m^s\right], \quad \Upsilon_i^c = \sigma_k\, \mathbf{E}\left(\cos(\omega_i^T x)x\right), \quad \Upsilon_i^s = \sigma_k\, \mathbf{E}\left(\cos(\omega_i^T x)x\right),$$

where $i = 1, \ldots, m$. We state the following proposition to compute each column in Υ consisting of expectations of sinusoidal functions and inputs.

Proposition 2. *The expectation of the multiplication of sinusoids and linear functions over multivariate Gaussian distributions: $x \sim \mathcal{N}(\mu, \Sigma)$, can be computed analytically:*

$$\mathbf{E}\left(\cos(\omega^T x)x\right) = \left(\mathbf{E}\cos(\omega^T x)\right)\mu - (\mathbf{E}(\sin(\omega^T x))\Sigma\omega, \qquad \mathbf{E}\left(\sin(\omega^T x)x\right) = \left(\mathbf{E}\sin(\omega^T x)\right)\mu + \left(\mathbf{E}\cos(\omega^T x)\right)\Sigma\omega,$$

where the right-hand-side expectations have analytical expressions (Proposition 1).

The proof is omitted. Next, the result is extended to $\mathbf{E}\left(x\cos(\omega^T x)\right)$, by setting a to consist of indicator vectors. Applying Proposition 1 and 2, we complete the derivation of $\mathbf{Cov}(x, y)$ in (11).

Remark: SSGP-EMM's computation complexity is $O\left(m^2k^2d^2\right)$, where m is the number of features, k is the output dimension, and d is the input dimension. Compared to the multivariate moment-matching approach for GPs (GP-EMM) [8, 10] with $O\left(n^2k^2d^2\right)$ time complexity, SSGP-EMM is more efficient when $m \ll n$. Moreover, our approach is applicable to any positive-definite continuous shift-invariant kernel with different spectral densities, while previous approaches like GP-EMM [10] are only derived for squared exponential (SE) kernels [1].

Method	SSGP-EMM	GP-EMM
Time	$O(m^2k^2d^2)$	$O(n^2k^2d^2)$
Applicable kernels	continuous shift-invariant kernels	SE kernels

Table 1: Comparison of our proposed methods and GP-EMM [8, 10] in computational complexity and generalizability.

4 Applications to Bayesian Filtering and Predictive Control

We focus on the application of the proposed methods to Bayesian filtering and predictive control. We consider the following discrete-time nonlinear dynamical system:

$$x_{t+1} = f(x_t, u_t) + w_t, \;\; w_t \sim \mathcal{N}(0, \Sigma_w), \qquad y_t = g(x_t) + v_t, \;\; v_t \sim \mathcal{N}(0, \Sigma_v), \tag{12}$$

where $x_t \in \mathbf{R}^d$ is state, $u_t \in \mathbf{R}^r$ is control, $y_t \in \mathbf{R}^k$ is observation or measurement, $w_t \in \mathbf{R}^d$ is IID process noise, $v_t \in \mathbf{R}^k$ is IID measurement noise, and subscript t denotes discrete time index. We consider scenarios where f and g are unknown but a dataset $\mathcal{D} = \left(\{(x_t, u_t), x_{t+1}\}_{t=1}^{n-1}, \{x_t, y_t\}_{t=1}^{n}\right)$ is provided. The dynamics model $p(x_{t+1}|x_t, u_t)$ is learned using state transition pairs $\{(x_t, u_t), x_{t+1}\}_{t=1}^{n-1}$, and the observation model $p(y_t|x_t)$ is learned separately from state-observation pairs $\{x_t, y_t\}_{t=1}^{n}$.

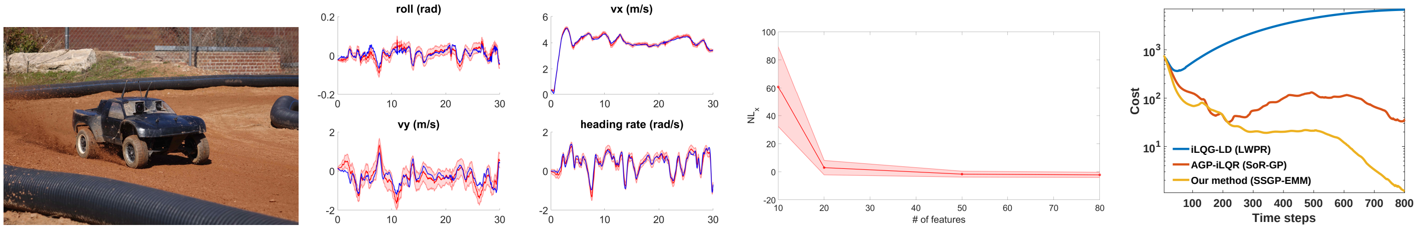
Bayesian filtering: The task of Bayesian filtering is to infer the posterior distribution of the current state of a dynamical system based on the current and past noisy observations, *i.e.*, finding $p(x_{t|t})$, where the notation $x_{t|s}$ denotes the random variable $x_t|y_0, \ldots, y_s$. Due to the Markov property of the process x, *i.e.*, $x_t|x_0, \ldots, x_{t-1} = x_t|x_{t-1}$, in Gauss-Markov models, $p(x_{t|t})$ can be computed

---

[1] Expressions for polynomial kernels can be derived similarly.

recursively through an alternating *prediction step* and *correction step*. We use the proposed SSGP-EMM to propagate full densities thorough the probabilistic dynamics and observation models instead of using linearization or finite-samples [9]. Our method is related to GP-ADF [6] which is based on GP-EMM [8, 10]. However, our method is more scalable and general as discussed in §3. In order to demonstrate the applicability of our method, we use a real-world state estimation task in high-speed autonomous driving on a dirt track (Figure 1a). The goal is estimating the vehicle's linear velocities, heading rate, and roll angle using only wheel speed sensors and ground truth samples generated by integrating GPS and IMU data. Filtered distributions using 80 features are shown in Figure 1b, and Figure 1c shows the negative log-likelihood of state NL_x for different number of features. Surprisingly, only a small number of features is necessary for satisfactory results.

Model Predictive Control: The goal of stochastic model predictive control (MPC) is to choose a control sequence that minimizes the expected cost, given a dynamics model and cost function. The main challenge of applying MPC in practice is efficient and accurate multi-step prediction due to the lack of explicit and accurate models. GPs have been used for dynamics modeling and prediction in control algorithms to cope with model uncertainty [4, 13]. However, these methods are restricted to off-line optimization due to the computational burden of GP inference. On the other hand, more efficient methods usually drop the uncertainty in the probabilstic model or input (1), e.g., iLQG-LD [12] uses Locally Weighted Projection Regression (LWPR), AGP-iLQR uses subset of regressors (SoR) approximation for GPs [1]. In this work, we combine our proposed SSGP-EMM and trajectory optimization (similar to [1, 12, 13]) for MPC. In addition, the SSGP dynamics model is updated incrementally. We demonstrate the performance of our algorithm using a simulated robotic arm manipulation task, in which our goal is tracking a moving target under dynamics variations. We use 1200 samples and 50 features for model learning and compare our method with iLQG-LD [12] and AGP-iLQR [1]. Results are shown in Figure (1d). Our method outperforms the other two because our multi-step predictions are more robust to model error. More precisely, the other methods do not propagate the full densities through probabilistic models of the dynamics, as we do here.



(a) Autonomous driving task (b) Filtered distribution vs. ground truth (c) NL_x vs. # of features (d) Predictive control cost

Figure 1: (a) High-speed autonomous driving task. (b) All state trajectories, blue lines are the ground truth (30 seconds continuous driving), red areas are filtered distributions. In (c), red is the mean and variance of the negative log-likelihood NL_x for the 1200 filtering steps (30 seconds driving). (d) The total trajectory cost comparison for iLQG-LD [12], AGP-iLQR [1] and our method.

5 Conclusion

We introduced an analytic moment-based approach to *prediction under uncertainty* in sparse spectrum Gaussian processes (SSGPs). Compared to its full GP counterpart [8, 10], our method is more general in terms of the choice of kernels, and is more scalable thanks to the sparse spectrum representation (see Table 1). Although we adopt the name SSGP, our proposed method is not tied to specific model learning methods such as linear Bayesian regression [11]. Our method is directly applicable to many sequential prediction and decision-making problems that involve uncertain dynamical systems. We demonstrated the performance of our method in Bayesian filtering and predictive control tasks using both real-world and simulated experiments.

References

[1] J. Boedecker, JT. Springenberg, J. Wulfing, and M. Riedmiller. Approximate real-time optimal control based on sparse Gaussian process models. In *ADPRL 2014*.

[2] J. Quinonero Candela, A. Girard, J. Larsen, and C. E. Rasmussen. Propagation of uncertainty in Bayesian kernel models-application to multiple-step ahead forecasting. 2003.

[3] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina F Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In *NIPS*, 2014.

[4] M. Deisenroth, D. Fox, and C. Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE PAMI*, 27:75–90, 2015.

[5] M. Deisenroth, R. Turner, MF Huber, UD Hanebeck, and CE Rasmussen. Robust filtering and smoothing with gaussian processes. *IEEE Trans. on Automatic Control*, 2012.

[6] Marc Peter Deisenroth, Marco F Huber, and Uwe D Hanebeck. Analytic moment-based gaussian process filtering. In *ICML*, 2009.

[7] A. Gijsberts and G. Metta. Real-time model learning using incremental sparse spectrum Gaussian process regression. *Neural Networks*, 41:59–69, 2013.

[8] A. Girard, C.E. Rasmussen, J. Quinonero-Candela, and R. Murray-Smith. Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting. In *NIPS*, 2003.

[9] J. Ko and D. Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009.

[10] Malte Kuss. *Gaussian process models for robust regression, classification, and reinforcement learning*. PhD thesis, Technische Universität, 2006.

[11] M. Lázaro-Gredilla, J. Quiñonero-Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal. Sparse spectrum gaussian process regression. *JMLR*, 99:1865–1881, 2010.

[12] D. Mitrovic, S. Klanke, and S. Vijayakumar. Adaptive optimal feedback control with learned internal dynamics models. 2010.

[13] Y. Pan and E. Theodorou. Probabilistic differential dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1907–1915, 2014.

[14] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2007.

[15] C. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. In *NIPS*, volume 4, page 1, 2004.

[16] C.K.I Williams and C.E. Rasmussen. *Gaussian processes for machine learning*. MIT Press, 2006.

The neural mechanisms of worse than expected prediction errors

Jessica Mollick[*]
Department of Psychology and Neuroscience
University of Colorado, Boulder
Boulder, CO 80309-0345
eCortex, Inc
Boulder, CO

Wolfgang Pauli
Computational and Neural Systems Program
Division of Humanities and Social Sciences
California Institute of Technology
Pasadena, CA 91125
`pauli@caltech.edu`

Luke Chang
Dartmouth College
Psychological and Brain Sciences
Dartmouth College
Hanover, NH
`luke.j.chang@dartmouth.edu`

Tor Wager
Department of Psychology and Neuroscience
University of Colorado, Boulder
Boulder, CO 80309-0345
`tor.wager@colorado.edu`

Randall O'Reilly
Department of Psychology and Neuroscience
University of Colorado, Boulder
Boulder, CO 80309-0345
`randy.oreilly@colorado.edu`

Abstract

Learning about conditioned inhibitors, which predict omission of outcome delivery, has been relatively understudied compared to learning about reward predictors. Reward omissions lead to inhibition of dopamine neurons, driven by the lateral habenula, an important region that is also involved in learning about predictors of punishment. How could a conditioned inhibitor, which has no primary punishment association, learn to drive this dopamine dip signal? We show that the temporal-differences algorithm can account for learning negative associations for a conditioned inhibitor, and used this model to construct regressors for an fMRI conditioned inhibition experiment ran with 19 subjects. We found neural correlates of a prediction error for a CS in the ventral tegmental area, as well as value signals in the substantia nigra/VTA, along with the amygdala and caudate and pallidum regions of the basal ganglia. We also discuss a biologically based artificial neural network model called the PVLV (Primary Value, Learned Value) model, that can learn conditioned inhibition, and proposes that amygdala circuitry is involved in controlling dopamine firing for conditioned stimuli, while ventral striatal circuitry communicates with the lateral habenula to control dopamine firing for unconditioned stimuli. This model's specification of the excitatory and inhibitory inputs to dopamine neurons in the different experimental conditions can be used to interpret fMRI signals in dopamine regions. This focus on worse than expected prediction errors can also be applied to understanding the learning about painful stimuli, and how disorders like depression and chronic pain involve distortions in these learning mechanisms leading to persistent negative predictions.

Keywords: learning, conditioned inhibition, reward, prediction error, value, habenula, aversive, neural network

Acknowledgements

This research was funded by NIH IBSC center 1-P50-MH079485, and NIH R01GM109996.

---

[*] https://grey.colorado.edu/CompCogNeuro/index.php/User:Jessica.mollick

1 Introduction

While the field of reinforcement learning has generally focused on the role of reward prediction errors in training reward expectations, there has been less of a focus on the learning for things that predict rewards will not occur, or **inhibitors**. In conditioned inhibition, a CS+ associated with reward is presented simultaneously with a conditioned inhibitor, which causes the expected reward not to occur. Computationally, inhibition is an interesting problem, because it relies on the negative prediction error driven by the expectation of the CS+ creating a negative value prediction for the conditioned inhibitor, even though it has never been paired with an aversive stimulus. Dopamine neurons respond to the inhibitor with an inhibition, the same pattern of dopamine release in the substantia nigra as an aversive stimulus [11, 8]. Intriguingly, recent research has shown that this dopamine inhibition, or dip, is driven by a brain region called the lateral habenula, which has been found to be activated during aversive processing and reward omissions [5]. After inhibition has been acquired the inhibitor can pass the summation test, meaning that the conditioned response to the CS paired with the inhibitor is less than to the CS alone [1].

2 Materials and Methods

In the present study, 19 participants participated in a Pavlovian conditioning paradigm during fMRI imaging, in which one fractal stimulus was associated with reward (orange juice) 75% of the time, while another fractal was associated with a neutral outcome (artificial saliva). In the first session of 48 trials, participants were exposed to the initial CS - US contingencies. In the second session of 180 trials, the CS+ was paired with another stimulus, the inhibitor, in 22 percent of trials. Presentation of the inhibitor was deterministically associated with the presentation of the neutral solution, negating the reward prediction by the CS+. In the remaining trials, participants continued to experience the initial CS - US paring, in order to keep the reward association from being extinguished by the conditioned inhibition procedure. In the last block, we run an inhibition test where we pair the inhibitor with an unexpected juice reward to see if a prediction error results due to the negative association.

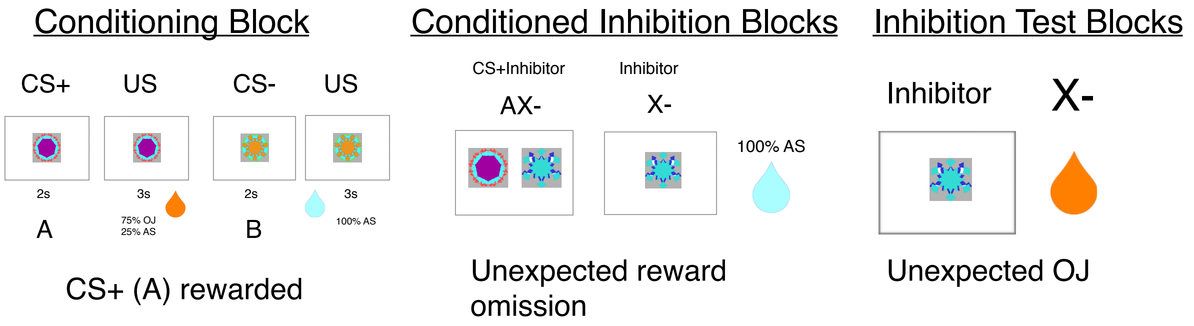


Figure 1: Experimental design: The CS+ is paired with an orange juice reward while a control CS- is paired with neutral solution. In the conditioned inhibition block, the CS+ is paired with an inhibitor which leads to reward omission.

3 Temporal-differences learning model

We simulated a TD learning model for the conditioned inhibition study. To simplify the learning problem, we specified three states in the experiment, the fixation cross, the onset of CS, and the onset of outcome delivery. Each CS had a different state value for the onset of that CS and the state immediately before US onset. The presentation of the reward occured in the state following the US onset and was coded as r=1 and the neutral solution as r=0. For a conjunctive stimulus, such as the CS+ (A) paired with the inhibitor (X), we simply combine the values for each stimulus included to lookup the overall value of that state.

$$V(AX) = V(A) + V(X) \tag{1}$$

Following the TD algorithm [10], a reward prediction error (RPE) was calculated by looking forward to the next state, and calculating the difference between the predicted value of the next state and the current state value, adding the value of the reward in the next state to the error calculation.

$$\delta = R(s_{t+1}) + \gamma V(s_{t+1}) - V(s_t) \tag{2}$$

Once the RPE from looking forward to the next state is calculated, it is used to update the value of the current state.

$$V(s_t) = V(s_t) + \delta \times \alpha \tag{3}$$

Similarly, when the value of a conjunctive stimulus was updated with a prediction error, this was split equally among the two stimuli to update their values, as was done in a previous TD model of conditioned inhibition [9]. Once the RPE is calculated, we update the value of the current state. For a conjunctive stimulus, we split the prediction error equally among the two stimuli to update their values.

$$V(AX) = V(A) + \alpha \times \frac{\delta}{2} + V(X) + \alpha \times \frac{\delta}{2} \tag{4}$$

We used the TD model to generate value and RPE estimates for each stimulus in the conditioned inhibition task, with a learning rate of 0.03, based on learning rates used in similar studies [6]. If conditioned inhibition occurred, the conditioned inhibitor (X), acquires a negative value, while the CS+ maintains a positive value throughout the conditioned inhibition procedure. Figure 2A shows the value for each CS as evidence that the TD model learned a positive value for the CS+ (A), but a negative association for the inhibitor (X). While our initial experiment did not include any behavioral measures of learning, in a second follow-up conditioned inhibition experiment, we observed a significantly lower rating of the level of reward expectation for the CS+ paired with the inhibitor compared to the CS+ alone, a sign that the conditioned inhibition worked and the inhibitor was able to reduce reward expectation.

3.1 Temporal Differences Model Imaging Results

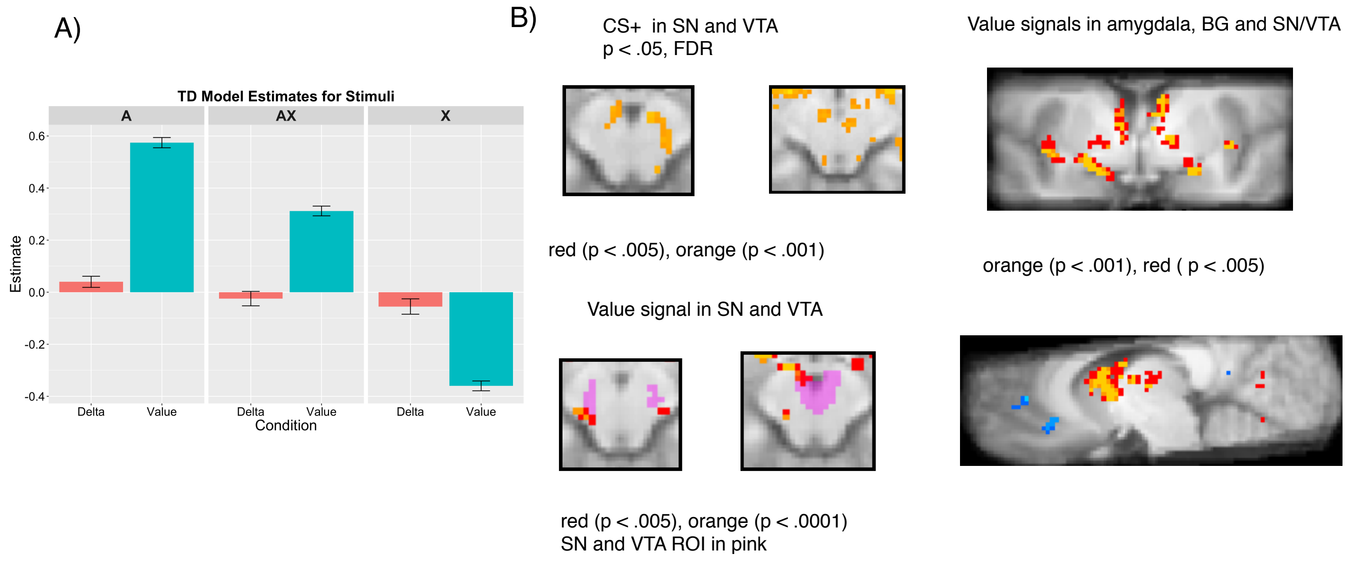


Figure 2:
A: Shows that the TD model learned a positive value for the CS+ (A) and a negative value for the inhibitor (X). B: There was a significant RPE in the VTA for the CS, and a significant value signal in the SN and VTA. Additionally, the whole-brain analysis showed significant value signals in the amygdala, basal ganglia and SN/VTA.

In order to investigate whether the valence of conditioned stimuli affected activity in anatomically defined ROIs of the ventral tegmental area (VTA) and the substantia nigra (SN), we first conducted a model-free analysis with CS type as independent variable. As predicted, we found significantly more activity in SN (t(18) = 2.99, p = .0079) and VTA (t(18) = 2.25, p = .0375) for the CS+ than the CS-, and activity in these regions for CS+ presentation (Fig. 2B). We predicted a reduction in SN/VTA activity for the conditioned inhibitor, accompanied by an increase in activity in the lateral habenula. Contrary to our predictions, habenula showed significant BOLD activity for the CS+, and marginal activity for the Inhibitor, and we did not see a significant reduction in SN/VTA activity for the inhibitor.

Next, we investigated which of the brain areas showed a pattern of activity that correlated with value and RPE estimates of the TD model. We found a significant delta signal during CS presentation in the VTA. There was also a significant value signal in the VTA and SN (Fig. 2B). Additionally, we saw a negative value signal in medial PFC. In keeping with our idea that different neural substrates may influence brain activity for the CS and US, we then looked separately at the RPE and value signals for the CS and the US. Our results showed that the VTA signal only reflected a RPE at the time of CS presentation (t(18) = 2.54, p = .0204), but not at the US. Additionally, the value signal in the amygdala seemed to be related to US presentation (t(18) = 4.40, p = .0003), but not at the time of the CS.

If activity for the CS increased monotonically according to RPE strength, then we would predict a positive linear slope for the CS RPE. As an additional validation of this RPE signal, we split all of the CS and US trials into four different bins with equal numbers of trials per bin for the different levels of RPE. However, the trials showed a nonlinear slope, with SN and VTA activity increasing for both positive and negative prediction errors, indicating an unsigned prediction error. If the inhibitor to acquire a negative association, there should be a prediction error during the inhibition test when the inhibitor is paired with a reward, resulting in activity in dopamine regions. However, an analysis looking at the mean activity in the VTA, SN or Accumbens ROIs for the inhibitor followed by a reward during this inhibition test did not show significant activity.

4 Biological modeling: PVLV model

While the TD model proposes a unitary mechanism to underlie the calculation of RPEs at in response CS and US onset, it is well established that different neural mechanisms control the dopamine firing for CS and US onset. This is the central tenet of the biologically-inspired PVLV neural network model [3], which proposes that the amygdala drives dopamine bursts to the CS, while the ventral striatum controls dopamine firing to the US, through circuitry including the lateral habenula. In particular, the model hypothesizes that the lateral habenula drives the dopamine dip for reward omissions, which trains up a negative association for the conditioned inhibitor. We therefore investigated whether any of these brain areas showed a pattern of BOLD responses that correlated with internal variables of the PVLV model. We found that the model learned conditioned inhibition and reproduces results of a previous study, which showed an inhibition of dopamine neurons in response to the inhibitor (X) [11]. In a preliminary analysis, we used this computational model to look at the different brain regions that contribute to the dopamine signal in this experiment and analyze the correlations between different excitatory and inhibitory inputs from the model and fMRI BOLD activity in dopamine regions, drawing on research showing that fMRI activity may be more related to synaptic input than activation of task-related neurons [4]. Strikingly, our results show that the summed measure of the model's estimated excitatory and inhibitory inputs to the SN/VTA from the computational model resembled the activity in these regions better than estimated dopamine neuron activation. Future work will involve correlating the excitatory and inhibitory input for the SN/VTA region with fMRI activity, to see whether excitatory or inhibitory input better explains BOLD signal than predicted neural activation of dopamine neurons.

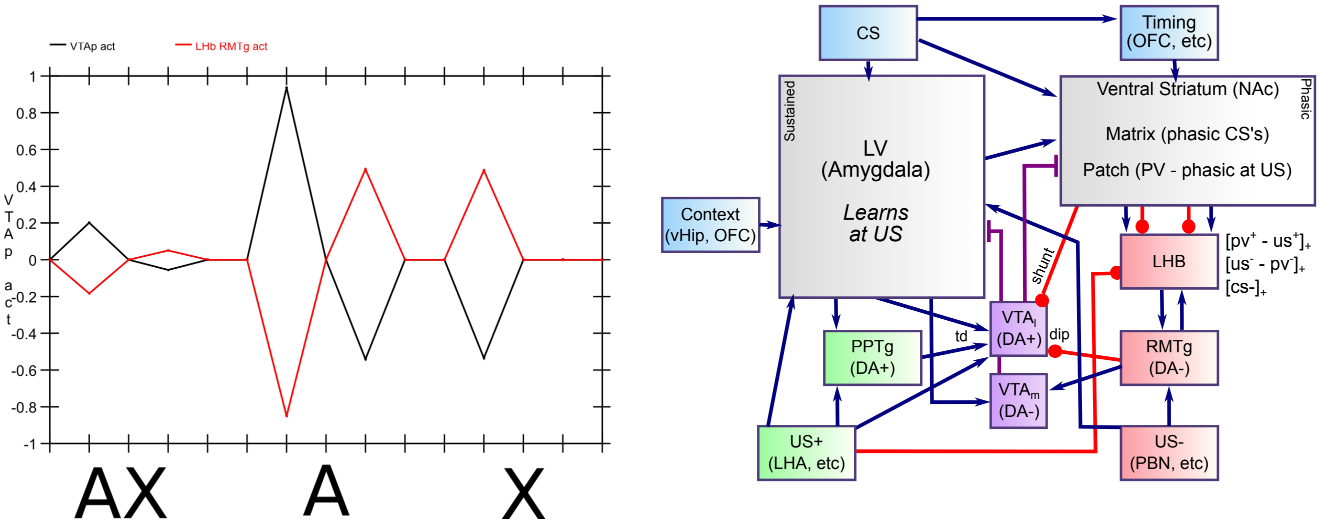


Figure 3: Results from the PVLV model for different types of conditioned stimuli, reproducing the dopamine dip (shown in black) to the conditioned inhibitor (X) shown in the conditioned inhibition study, as well as the burst to the CS+ (A). The PVLV model, shown on the right, uses one system for learning about the CS (the LV system), and another (PV) system for learning about the US in the ventral striatum, which communicates with the lateral habenula to drive dopamine dips for worse than expected rewards.

5 Discussion

Focusing on the mechanisms involved in learning about conditioned inhibitors is important because it relies on an understanding of the neural mechanisms that drive negative, or worse than expected, prediction errors. The lateral habenula, which also seems to play an important role in depression, seems to be an important region for driving these signals. We replicated prior work showing delta and value signals in dopamine regions [2, 7], while also showing that the lateral habenula showed a greater negative correlation with SN/VTA than the other stimuli in the experiment. Along with the TD model, I discussed applications of a biologically based computational model, the PVLV model, to the conditioned inhibition task, and how this model can be used to look at the correlation between fMRI signals and different inputs to the SN/VTA. This work can help clarify interpretation of dopamine signals observed in fMRI studies, and provides hypotheses about the different neural inputs that drive dopamine firing for CSes and USes.

References

[1] R a Rescorla. Conditioned inhibition of fear resulting from negative CS-US contingencies. *Journal of comparative and physiological psychology*, 67(4):504–9, April 1969.

[2] Kimberlee D'Ardenne, Samuel M. McClure, Leigh E. Nystrom, and Jonathan D. Cohen. BOLD Responses Reflecting Dopaminergic Signals in the Human Ventral Tegmental Area. *Science*, 319(5867):1264–1267, February 2008.

[3] Thomas E. Hazy, Jessica Mollick, Ananta Nair, Kai Krueger, Prescott Mackie, Seth Herd, and Randall C. O'Reilly. Integrating primary aversive outcomes into a neuro-computational model of the phasic dopamine signaling system: A critical role for valence-congruence. (in prep).

[4] Nikos K. Logothetis. What we can do and what we cannot do with fMRI. *Nature*, 453(7197):869–878, June 2008.

[5] Masayuki Matsumoto and Okihide Hikosaka. Representation of negative motivational value in the primate lateral habenula. *Nature Neuroscience*, 12(1):77–84, January 2009.

[6] John P O'Doherty, Peter Dayan, Karl Friston, Hugo Critchley, and Raymond J Dolan. Temporal difference models and reward-related learning in the human brain. *Neuron*, 38:329–337, 2003.

[7] Wolfgang M. Pauli, Tobias Larsen, Sven Collette, Julian M. Tyszka, Ben Seymour, and John P. O'Doherty. Distinct Contributions of Ventromedial and Dorsolateral Subregions of the Human Substantia Nigra to Appetitive and Aversive Learning. *The Journal of Neuroscience*, 35(42):14220–14233, October 2015.

[8] Wolfram Schultz. Behavioral dopamine signals. *Trends in Neurosciences*, 30(5):203–210, May 2007.

[9] Richard S Sutton and Andrew G Barto. A temporal-difference model of classical conditioning. In *Proceedings of the ninth annual conference of the cognitive science society*, pages 355–378. Seattle, WA, 1987.

[10] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 1998. *A Bradford Book*, 1998.

[11] Philippe N Tobler, Anthony Dickinson, and Wolfram Schultz. Coding of predicted reward omission by dopamine neurons in a conditioned inhibition paradigm. *The Journal of neuroscience*, 23(32):10402–10, December 2003.

Strategies of Observational Reinforcement Learning

Ida Selbing
Dept. of Clinical Neuroscience
Karolinska Institutet
`ida.selbing@ki.se`

Andreas Olsson
Dept. of Clinical Neuroscience
Karolinska Institutet
`andreas.olsson@ki.se`

Abstract

Humans learn about their environment both from own experiences, and through observing others, here referred to as “demonstrators”. One form of observational learning is learning through observation of demonstrators’ choices and the ensuing outcome to update expectations of the outcome of choices, i.e. “observational reinforcement learning”. We have previously shown that participants learning a simple two choice task preferably learn through observational reinforcement learning rather than through copying. In addition, empirical results from our lab suggest that participants tended to value observational information higher if the demonstrator’s level of performance is described as high rather than low. To investigate how the level of performance of the demonstrator effects the efficiency of observational reinforcement learning, we simulated observational learning from demonstrators with three levels of knowledge of the choice-outcome contingencies of the environment they act in: None, Some and Full. Simulations were carried out for choice tasks with probabilistic outcomes varying in size: 2, 3 and 10 choices. The results from these simulations show that for the 2-choice task, observational reinforcement learning is more efficient the less knowledge the demonstrator has and the more random its choices are. However, as the size of the task increased, the more valuable was the observational information from a demonstrator with high knowledge, specifically at the start of the observer’s learning. These results provide support for the rationality of an overall strategy to pay more attention to observational information from demonstrators that are high performers. However, the simulations also show that this strategy is not always optimal, and that adhering to it may in many cases result in suboptimal performance.

Keywords: Reinforcement Learning, Observational Learning, Strategy selection

Acknowledgements

This research was supported by an Independent Starting Grant (284366; Emotional Learning in Social Interaction project) from the European Research Council and the Knut and Alice Wallenberg Foundation (KAW 2014.0237) to A.O.

1 Introduction

Learning through observation of others is a common and efficient way of learning about the world. Learning can occur through observation of the outcome of other's behaviors, here referred to as observational reinforcement learning but also through copying of others' behaviors. A study on observational learning using a probabilistic two-choice task conducted in our lab showed that participants learned through observational reinforcement learning rather than copying when information of the outcome of an observed demonstrator's choice was available [1]. Participants' level of performance did not differ as a function of the demonstrators' performance. In this first study we gave no explicit information about the performance of the demonstrators. Following this, we have in two separate studies (in prep.) showed that when demonstrators are labelled as high, as compared to low in performance, participants are better at learning from these demonstrators, even when they make random choices. Results from analyses of pupil dilation responses suggest that this difference in performance can be explained, at least partly, by a difference in the attention directed towards the observational information (i.e. the demonstrator's choice and the ensuing outcome). Our results appear contradictory if we attempt to interpret them within a framework of attention as a utility maximizing system [2]; participants were able to learn equally well from a demonstrator that behaved randomly as from one that performed relatively well but nonetheless appeared to follow the strategy to sample information from the demonstrator they believed to be the best when given the opportunity. To investigate the effect of the demonstrator's level of performance on observational learning we have simulated observational learning from demonstrators make choices based on three levels of knowledge about the task's outcome contingencies: None (no knowledge), Some (some knowledge) and Full (full knowledge). We hypothesize that the efficiency of observational learning depends on the interaction between demonstrator behavior and task structure. Therefore, simulations were carried out over probabilistic n-choice tasks with varying number of choices. In addition to providing guidance for us to interpret our experimental data this work is a first step in uncovering optimal strategies for observational reinforcement learning in various environments. This further allows us to generate testable predictions of human behavior.

2 Simulation of Observational Reinforcement Learning

The simulations presented here are based on a simple model of observational learning where one run consists of an observer observing and learning from one demonstrator. An observer makes choices based on its learning history, receives outcomes but does not learn from its own outcome. Each combination of the demonstrator's level of knowledge and number of available choices in the task is simulated over 10000 runs of 150 trials.

2.1 The Task

Each task is constructed as a probabilistic n-choice task were the outcome of each choice is drawn from a normal distribution with a fixed mean and a standard deviation of 10. The means are randomly generated, centered around 0 and constrained so that the average distance between the n means is equal to one standard deviation (i.e. 10). Three task sizes are simulated; 2, 3 and 10 choices, see Figure 1 for illustrative examples. Note that for the 2-choice task the constraints imposed on the outcome distributions only allows for outcome distributions to be centered at -5 and 5.

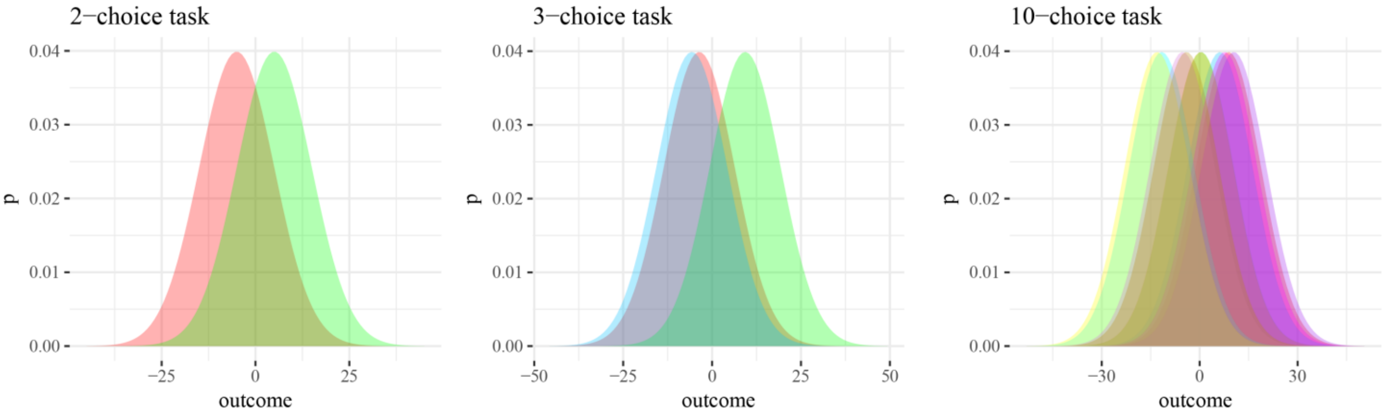


Figure 1. Examples of outcome distributions for tasks consisting of 2, 3, and 10 choices.

2.2 The Demonstrators

The demonstrators select among the choices *i* in the task according to the softmax function based on a set of expected choice values, Q_n, with β fixed to 1:

$$P_i(t) = \frac{\exp(Q_i(t)/\beta)}{\sum_{k=1}^{n}\exp(Q_k(t)/\beta)} \qquad (1)$$

To investigate the effect of the demonstrator's performance on the observer's performance we varied the demonstrators' knowledge of the choice-outcome contingencies by varying the expected choice values Q_n.

- For knowledge condition None, Q_n was fixed to 0 for all n, analogous to the demonstrator having no knowledge about the contingencies and no ability to differentiate between choices.
- For knowledge condition Full, Q_n was fixed to the means of the outcome distributions so that the demonstrator's expected choice values accurately reflected the real expected values of the choices, analogous to perfect knowledge of the outcome contingencies.
- For knowledge condition Some, we initially set Q_n to 0 for all choices and then allowed the demonstrators to learn about the task prior to being observed. Demonstrators learned by making random choices and updating Q_n for the selected choice *a* using the received outcomes according to the Q-learning algorithm [3] with the learning rate α set to 0.3:

  $$Q_a(t+1) = Q_a(t) + \alpha * (outcome - Q_a(t)) \qquad (2)$$

  The expected values were fixed and the demonstrator stopped learning when the sum of the absolute differences between the demonstrator's expected values and the means of the outcome distribution was smaller than 90% of the initial difference when Q_n was 0 for all choices n. This is analogous to the demonstrators having some knowledge of the outcome contingencies. Observers only observed the demonstrators' choices after the expected values were fixed.

Importantly, during observation, none of the demonstrators learned or adjusted their expected values in response to outcomes (see Figure 2).

2.3 The Observers

Observers learned through observing the demonstrator's choices and the ensuing outcomes according to equation (2). Choices are made according to equation (1). To be able to clearly distinguish the effect of demonstrator behavior on performance, observers could not learn from their own choices.

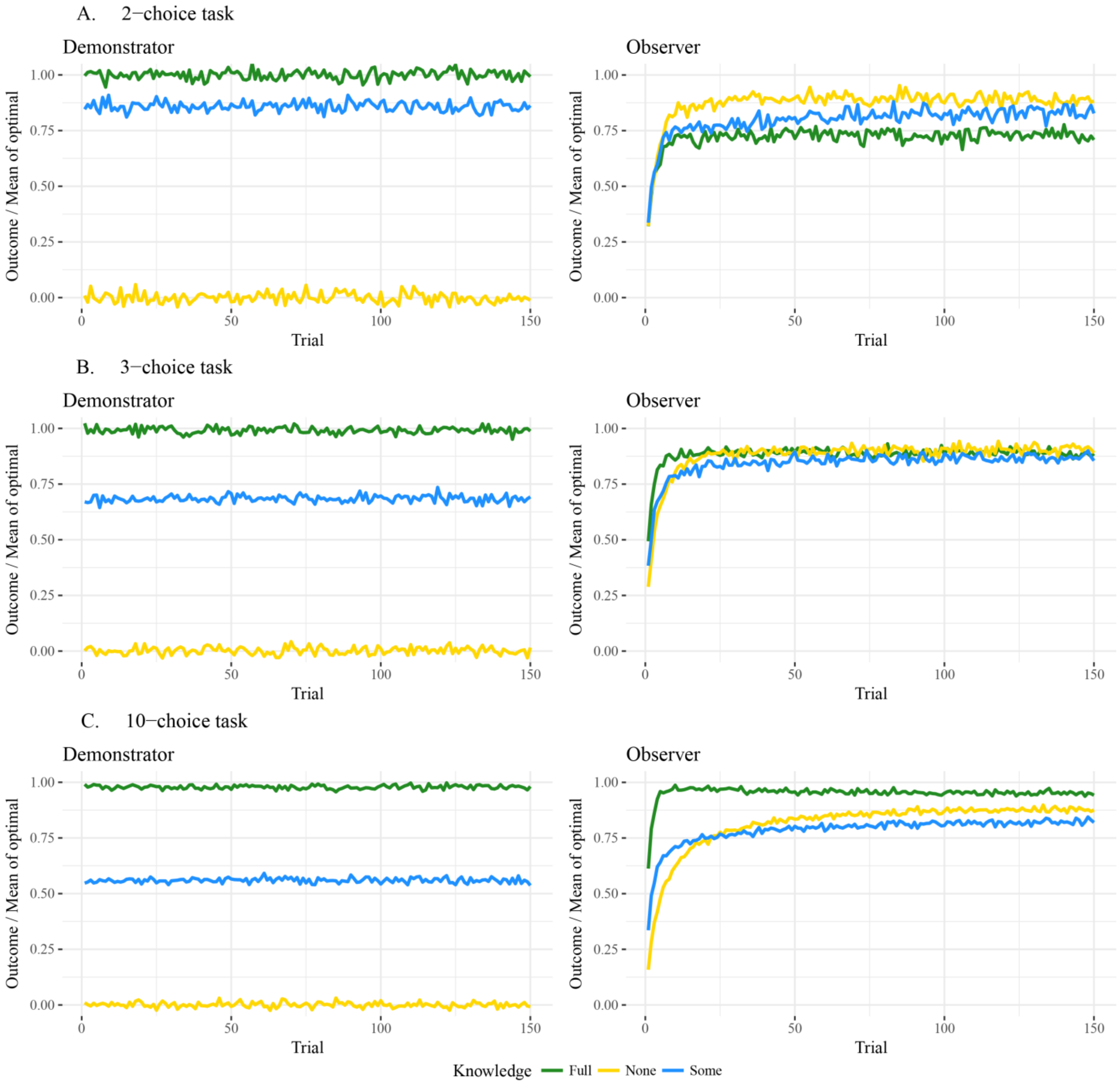


Figure 2: Mean performance of demonstrators and observers across trials. A. 2-choice tasks. Observation of a demonstrator in knowledge condition None is the most optimal already from trial 2. B. 3-choice tasks. Initially, observation of a demonstrator in knowledge condition Full is the most optimal. C. 10-choice tasks. Observation of a demonstrator in knowledge condition Full is the most optimal through all 150 trials.

3 Results

To be able to easily compare performance between runs, performance is calculated as the outcome at each trial divided by the mean of the outcome of the optimal choice. We show here the mean performance per trial and knowledge condition for both the demonstrator and observer for the three task sizes. The results of the simulation of the 2-choice task show that for these types of tasks, observation of a demonstrator in knowledge condition None is the best, although the differences are small. Interestingly, when the size of the task is increased we see that the value of observing the demonstrators shift over trials; initially, higher levels of the demonstrators' knowledge leads to higher performance in the observer while this relationship changes over time. The advantage of observing a demonstrator with no knowledge of the choice-outcome contingencies is that such as demonstrator will explore the full choice-space and thus quicker give the observer better knowledge of all choices. The advantage of observing a demonstrator with knowledge of the choice-outcome contingencies is that it allows the observer to quickly discriminate between the most optimal choices and the rest.

4 Discussion

We have shown here that observational reinforcement learning of a 2-choice task is more efficient the less knowledge the demonstrator has and the more random its choices are. However, increasing the size of the task makes observational information from knowledgeable demonstrators more valuable, although this effect is more pronounced at the beginning and then changes over time. Our results provide support for the usefulness of an overall strategy to pay more attention to observational information from demonstrators believed to be high performers even though it is clear that such a strategy is not always optimal. Under-sampling of failure has for instance been linked to false beliefs in the nature of effective management [4]. Our results rely on a very simple model of observational reinforcement learning. Future work will focus on refining the learning models. Our findings also allow us to generate testable predictions of human behavior.

References

[1] I. Selbing, B. Lindström, and A. Olsson, "Demonstrator skill modulates observational aversive learning.," *Cognition*, vol. 133, no. 1, pp. 128–39, Oct. 2014.

[2] J. Gottlieb, "Attention, learning, and the value of information.," *Neuron*, vol. 76, no. 2, pp. 281–95, Oct. 2012.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: The MIT Press, 1998.

[4] J. Denrell, "Vicarious Learning, Undersampling of Failure, and the Myths of Management," *Organ. Sci.*, vol. 14, no. 3, pp. 227–243, 2003.

Shaping Model-Free Reinforcement Learning with Model-Based Pseudorewards

Paul M. Krueger
Dept. of Psychology, UC Berkeley
pmk@berkeley.edu

Thomas L. Griffiths
Dept. of Psychology, UC Berkeley
tom_griffiths@berkeley.edu

Abstract

Model-free (MF) and model-based (MB) reinforcement learning (RL) have provided a successful framework for understanding both human behavior and neural data. These two systems are usually thought to compete for control of behavior. However, it has also been proposed that they can be integrated in a cooperative manner. For example, the Dyna algorithm uses MB replay of past experience to train the MF system, and has inspired research examining whether human learners do something similar. Here we introduce Model-Based Pseudoreward Approximation (MBPA), an approach that links MF and MB learning in a new way: via the reward function. Given a model of the learning environment, dynamic programming is used to iteratively estimate state values that monotonically converge to the state values under the optimal decision policy. Pseudorewards are calculated from these values and used to shape the reward function of a MF learner in a way that is guaranteed not to change the optimal policy. We show experimentally that MBPA offers computational advantages over Dyna. It also offers a new way to think about integrating MF and MB RL: that our knowledge of the world doesn't just provide a source of simulated experience for training our instincts, but that it shapes the rewards that those instincts latch onto. MBPA should motivate new hypotheses to test experimentally in human cognition and neural data.

Keywords: model-free reinforcement learning; model-based reinforcement learning; reward shaping

1 Introduction

The problem of learning from environmental rewards has been studied extensively in both psychology and artificial intelligence research. Both fields have explored two different approaches to solving this problem, known as model-free (MF) and model-based (MB) reinforcement learning (RL). MF learning relies on direct trial-and-error interaction with the environment (Sutton, Barto, & Williams, 1992), while MB learning leverages knowledge about the causal structure of the environment (Barto, Bradtke, & Singh, 1995). Historically, animal psychologists viewed these two systems as distinct and competing hypotheses, with behaviorists arguing in favor of reflexive, MF learning based on stimulus-response associations (Thorndike, 1933), and Tolman and others positing an internal representation of the environment, or "cognitive map" (Tolman, 1948).

Nowadays, while behavioral and neural data indicate that human learning relies on both systems (Daw, Niv, & Dayan, 2005; Gläscher, Daw, Dayan, & O'Doherty, 2010; Dayan & Berridge, 2014), it is typically assumed that they compete for control of behavior. However, it is also possible for them to cooperate. The Dyna architecture achieves such cooperation by integrating MF learning with MB planning (Sutton, 1991). In Dyna, as MF learning occurs, transitions between states of the environment and the resulting rewards are stored in a model. That model is used to replay these past experiences, using them to further train MF state-action values.

Here we introduce a new method for cooperative interaction between MF and MB learning. The MB system generates pseudorewards that shape the reward function used in MF learning. According to the *shaping theorem*, conditions exist under which the optimal decision policy will remain invariant to such modifications of the reward function, opening the possibility that pseudorewards can be used to guide agents toward optimal behavior (Ng, Harada, & Russell, 1999). We show that pseudorewards can provide a link between MF and MB learning through modification of the reward function.

This method of cooperation between learning systems offers an appealing alternative to Dyna, both conceptually and practically. With Dyna, the MB replay of past experience suggests that planning (by internal simulation) is one way that different learning systems might be linked in human cognition. The method that we introduce offers an alternative approach, based on changing the reward function, which can be tested experimentally in humans. One way that this link may manifest in human cognition is through MB production of emotions that function as pseudorewards for MF learning. In addition to providing a new way to think about interaction between RL systems, our method also offers practical advantages over Dyna by learning in fewer steps and requiring less computation time.

We begin by reviewing the Dyna architecture for integrated MF and MB learning. We then introduce our method and the theoretical background on which it is based. We present an experiment which shows the effectiveness of our method and how it compares with Dyna.

2 Cooperative Reinforcement Learning

2.1 Markov Decision Processes

We describe sequential decision problems that can be modeled as a *Markov decision process (MDP)*, $M = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, and $\mathcal{P}(s, a, s')$ is the probability of transitioning to state s' when action a is selected in state s, $\mathcal{R}(s, a, s')$ is the corresponding reward, and γ is the discount on rewards (Sutton & Barto, 1998). A *policy*, π, is a mapping of states, $\mathcal{S}$, onto actions, $\mathcal{A}$: $\pi : \mathcal{S} \mapsto \mathcal{A}$. A *value function*, $V^\pi(s)$, is the expected discounted reward generated by following policy π beginning at state s: $V^\pi(s) = \sum_{s'} P_{\pi(s)}(s, a, s')(R_{\pi(s)}(s, a, s') + \gamma V^\pi(s'))$. An *optimal policy*, π^*, is a policy that maximizes the value function: $\pi^*(s) = \arg\max_a V^\pi(s)$.

2.2 Model-free Reinforcement Learning

RL is concerned with learning an effective policy from rewards alone. MF methods require no knowledge about the environment, and the agent learns which state-action pairs lead to reward through trial-and-error. One of the most common MF methods, which is employed throughout the experiments in this paper, is Q-learning (Sutton & Barto, 1998). When the agent takes action a from state s, leading to state s' and reward $R(s, a, s')$, a value $Q(s, a)$ is learned via the update $Q(s, a) \leftarrow Q(s, a) + \alpha(R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a))$, where α is the *learning rate* that determines how quickly the agent learns from new experience. The agent uses a *decision policy*, such as the ϵ-greedy policy which is used in our experiments. At each state s, with probability $1 - \epsilon$, the agent chooses the action $a \in \mathcal{A}$ with the highest value $Q(s, a)$. With probability ϵ it chooses an action uniformly at random.

2.3 Model-based Reinforcement Learning

Unlike MF RL, MB RL has knowledge of the environment in terms of the transition probabilities between states, $\mathcal{P}$, and the reward contingencies for state-action pairs, $\mathcal{R}$. One of the most common MB methods for finding an optimal policy

π^* is *dynamic programming* which calculates the value of state s under policy π according to the *Bellman equation*: $V^\pi(s) = R_{\pi(s)}(s, \pi(s), s') + \gamma \sum_{s'} P_{\pi(s)}(s, a, s') V^\pi(s'))$, and finds the value of each state $V^*(s)$ under the optimal policy π^* by recursively updating these values using the *Bellman optimality equation*: $V^*(s) = \max_a R_{\pi(s)}(s, a, s') + \gamma \sum_{s'} P_{\pi(s)}(s, a, s') V^*(s'))$.

2.4 Dyna

Dyna uses MF learning combined with a MB system that replays past experiences, which are used to train the MF system. After each real action taken in the environment, the model stores the state-action pair and reward received. It then randomly selects n past state-action pairs and replays them. These planned actions are used to update the MF system as if they were real actions. In Dyna-Q, the MF system uses one-step tabular Q-learning (which is what we use in our experiments). The number of simulated planning steps, n, is a parameter that can be set to any positive integer.

In addition to being a useful algorithm for integrating direct learning with indirect replay, Dyna has been proposed as a model of human cognition. Gershman et al. (2014) found behavioral evidence in humans consistent with a Dyna architecture. Participants performed a sequential decision task with separate learning phases that tested behavioral revaluation. When given either more time between phases or a smaller cognitive load, the magnitude of revaluation was larger, consistent with MB replay of past experience. There are also neurophysiological data that suggest Dyna-like cooperation between the two systems. Lansink et al. (2009) identified neurons in the hippocampus of rats encoding spatial location and neurons in the striatum encoding reward. During sleep, the activation of those hippocampal cells correlated with and proceeded activation of the same striatal cells that encoded the value of those locations.

3 Model-Based Pseudoreward Approximation

Dyna integrates MF and MB RL by simulating past experience. We now consider Model-Based Pseudoreward Approximation (MBPA), a different way to merge the two. Our method uses dynamic programming to approximate state values. These values are used to calculate pseudorewards according to the shaping theorem. By shaping the reward function, pseudorewards provide a link between MB planning and MF learning.

3.1 Pseudorewards and the shaping theorem

Pseudorewards offer a way of conferring extra information to an agent about the reward landscape. Essentially, a small reward is given to the MF agent whenever it takes an action that helps the agent move towards the goal. Instead of the agent receiving actual reward $R(s, a, s')$ when moving from state $s \to s'$, the agent receives an augmented reward $R'(s, a, s')$ where $R'(s, a, s') = R(s, a, s') + F(s, a, s')$.

Pseudorewards are defined using *shaping functions*, F. In Ng et al. (1999) conditions for which the optimal policy π^* remains invariant under a shaping function are developed. In particular, F necessarily must be a potential-based shaping function to possess this invariance property: $F(s, a, s') = \gamma\Phi(s') - \Phi(s)$, where Φ is a real-valued function, $\Phi : \mathcal{S} \to \mathbb{R}$. If the shaping function is not potential-based, it is possible that Q-learning will converge to a suboptimal solution. The simplest example of invariant pseudorewards uses the difference in optimal values between the agent's current state and next state: $F(s, a, s') = \gamma V^*(s') - V^*(s)$. This method is called the *optimal policy pseudoreward* – it encourages the agent to always move down the optimal path from its current state.

With optimal policy pseudorewards the agent can maximize long-term reward simply by taking the most rewarding action at each step. However, in real-world scenarios, it may be unrealistic for a human to have such complete information. Computing the optimal policy may require many iterations of the Bellman equation, or solving a linear program.

3.2 Approximating the value function

The optimal policy pseudorewards require knowing the value function under the optimal policy, but that may be costly to compute. Alternatively, the optimal value function can be approximated, requiring less computation. Bounded Real-Time Dynamic Programming is a planning algorithm that attains certain performance guarantees if its lower- and upper-bounded estimates of state values converge monotonically toward state values under the optimal policy (McMahan, Likhachev, & Gordon, 2005). Importantly, this monotonic convergence toward optimal values is guaranteed to occur if the lower and upper bounds are initialized properly. Here, we take advantage of this monotone property to calculate approximate state values using dynamic programming. Specifically, any number, n, of iterations of the Bellman equation can be used to approximate state values, and as n increases, state values converge toward optimal values. These values after n iterations are then used to approximate pseudorewards according to the shaping theorem. Thus, there is a tradeoff, determined by n, between the proximity of pseudorewards to their optimal values and the amount of computation.

3.3 Linking model-free and model-based RL with the reward function

Figure 1 provides a schematic illustration of MBPA, wherein dynamic programming is used to approximate pseudorewards, which in turn shape the reward function and policy of the MF agent. We are interested in describing situations in which humans already have a model of the environment and use this information to train their MF instincts. A model containing state-action pairs and reward contingencies is used to estimate state values using n iterations of the Bellman equation. These values are used to calculate pseudorewards according to the shaping theorem, which are added onto real rewards. In this way, the MF agent is guided by pseudorewards that are generated using MB RL. In the remainder of the paper we present an experiment focused on evaluating MBPA and comparing it to Dyna.

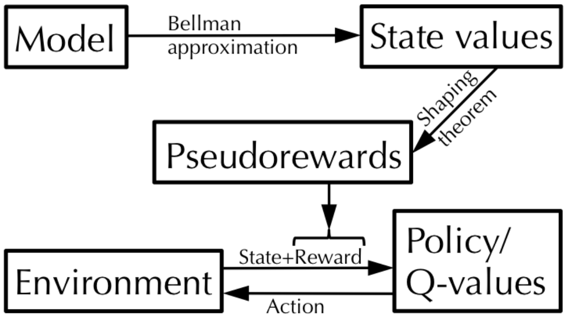


Figure 1: Model-Based Pseudoreward Approximation.

4 Maze learning experiment

4.1 Methods

Our experiment involved an agent learning in a maze environment (Sutton, 1991). The agent (a simple Q-learner), began each episode in the upper-left corner of a maze, and was rewarded one point for reaching the lower-right corner. The state space consisted of 121 locations, 50 of which were walls, and actions consisted of each of the four cardinal directions. The agent was trained for fifty episodes, with each episode ending when the goal was reached or 2,000 steps were taken. An ϵ-greedy decision policy was used with $\epsilon = 0.25$. Rewards were discounted with $\gamma = 0.95$, and therefor the value of each state is 0.95^d, where d is the minimum number of steps to the goal. We averaged 100 simulations.

Approximate pseudorewards Dynamic programing was used to approximate state values by iterating over the Bellman equation. In McMahan et al. (2005) conditions are defined under which initial state values will provably converge monotonically toward optimal values, but they note that in practice most reasonable initial values will achieve this monotonic convergence. Here, all states were initialized with a lower bound of zero and an upper bound of one, which in our environment is known to bound state values, and monotonic convergence was confirmed. The point at which each state reaches its optimal value is exactly equal to the minimum number of steps that state is from the goal, d. At each state, the pseudoreward for each action was calculated according to the shaping theorem as the difference between the value of the current state and the value of the next state given that action (which was deterministic). Either the lower-bound or the upper-bound of state values after n iterations of Bellman updates was used to approximate pseudorewards.

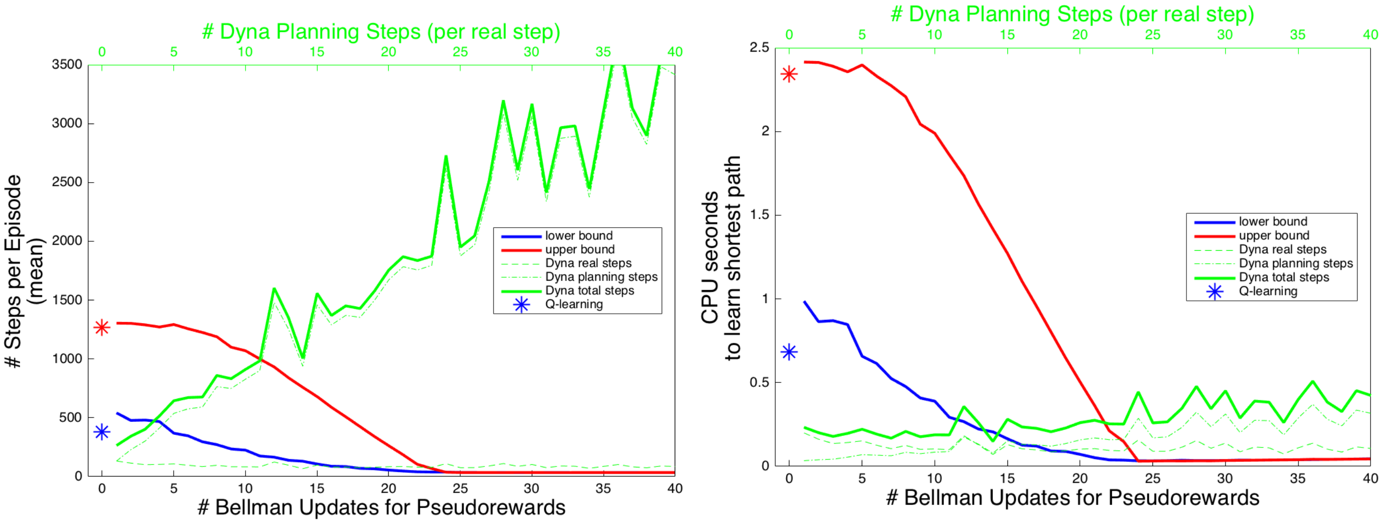
Trading off MF and MB computation The closer pseudorewards are to their optimal values, the easier the learning for the MF agent (to some precision). However, whereas Q-learning is simple and quick, the MB method of approximating state values is relatively slow and computationally costly. Therefore, we sought to understand the most efficient tradeoff between MB pseudoreward approximation and MF learning, in terms of CPU time required for each algorithm.

4.2 Results

Figure 2a shows the number of steps per episode needed to reach the goal, averaged across 50 episodes, as a function of the the number of Bellman updates used to approximate pseudorewards. As expected, learning is quicker when pseudorewards are closer to their optimal values. We also show performance of the Dyna agent as a function of the number of planning steps taken after each real step. While approximate pseudorewards are calculated just once using n iterations, the n planning steps used by Dyna are taken *after every single step of every episode*.

The number of real steps alone taken by the Dyna agent do not converge as low as the MBPA agent. With sufficiently precise pseudorewards, the MBPA agent, on the other hand, can learn the shortest path on the very first episode. Specifically, 24 Bellman updates are required for this, because the start state is 24 steps away from the goal state; after 24 iterations of the Bellman equation, optimal state values have propagated back from the goal state to the start state. Also shown are the number of steps taken by a Q-learning agent when state values are initialized to 0 (blue asterisk) or 1 (red asterisk).

Next, we calculated the actual time required to learn the shortest path. While the pseudoreward method may take fewer steps to reach the goal than Dyna, it does not necessarily mean that it is faster; planning steps (which use scalar operations to update Q-values) are about two orders of magnitude quicker than Bellman updates (which require matrix



(a) MBPA learns in fewer steps than Dyna.

(b) MBPA learns the shortest path more quickly than Dyna with maze learning.

multiplication). However, Figure 2b shows that MBPA is still faster than Dyna. The fastest learning occurs when 24 iterations of the Bellman equation are used; any more than this is unnecessary and the CPU time increases.

We also ran a second experiment in a mountain car environment and found a similar pattern of results, whereby MBPA learned in fewer steps and less CPU time than Dyna.

5 Discussion

We have introduced MBPA, a new method for cooperatively integrating MF and MB RL. This method relies on Bounded Real-Time Dynamic Programming to estimate state values that converge to values under the optimal policy. These approximate values are used to calculate pseudorewards according to the shaping theorem, such that the reward function is altered but the optimal policy is not. This modified reward function is used for MF learning. Our experiments demonstrate that this method performs comparably to and even better than Dyna, a popular cooperative RL algorithm.

We are interested in how already-learned models of the environment may be used to generate pseudorewards in human cognition, which is why MBPA inherits such knowledge (although it can also learn a model). Conversely, Dyna typically builds a model of the environment, but can also learn one. Future work should focus on evaluating our method empirically through behavioral and neural experimentation. In particular, it would be interesting to test whether certain emotions that are internally generated may function as pseudorewards in a manner consistent with MBPA.

References

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, *72*(1), 81–138.
Daw, N. D., Niv, Y., & Dayan, P. (2005). Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, *8*(12), 1704–1711.
Dayan, P., & Berridge, K. C. (2014). Model-based and model-free pavlovian reward learning: revaluation, revision, and revelation. *Cognitive, Affective, & Behavioral Neuroscience*, *14*(2), 473–492.
Gläscher, J., Daw, N., Dayan, P., & O'Doherty, J. P. (2010). States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron*, *66*(4), 585–595.
McMahan, H. B., Likhachev, M., & Gordon, G. J. (2005). Bounded real-time dynamic programming: Rtdp with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd international conference on machine learning* (pp. 569–576).
Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, *13*(1), 103–130.
Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml* (Vol. 99, pp. 278–287).
Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, *2*(4), 160–163.
Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press.
Sutton, R. S., Barto, A. G., & Williams, R. J. (1992). Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems*, *12*(2), 19–22.
Thorndike, E. L. (1933). A proof of the law of effect. *Science*.
Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological review*, *55*(4), 189.

Cross-Domain Transfer Learning using Target Apprentice

Girish Joshi
Coordinated Science Lab
Department of Agriculture & Biological Engineering
University of Illinois, Urbana-Champaign
Urbana, IL 61801
`girishj2@illinois.edu`

Girish Chowdhary [*]
Coordinated Science Lab
Department of Agriculture & Biological Engineering
University of Illinois, Urbana-Champaign
Urbana, IL 61801
`girishc@illinois.edu`

Abstract

In this paper we present a new approach to transfer learning in RL for cross domain tasks. Many of the available techniques, approach the transfer architecture as method of speeding up the target task RL. We propose to adapt and reuse directly, the mapped source task optimal policy in related target domains. We demonstrate that the optimal policy from a related source task can be near optimal in target domain provided an adaptive policy accounts for the model error between target and source. The main benefit of this policy augmentation is generalizing learning across multiple related domains without learning in target space. Our results show that this architecture leads to better sample efficiency in transfer as the sample complexity of target task learning is reduced to target apprentice learning.

Keywords: Reinforcement Learning, Transfer Learning, Apprentice Learning,
Inter-Task Mapping, Manifold Alignment

Acknowledgements

This work is supported by AFOSR FA9550-15-1-0146

---

[*] Asst Professor, Agricultural and Biological Engineering, UIUC (http://daslab.illinois.edu/drupal/home)

1 Introduction

Reinforcement Learning (RL) is a machine learning paradigm, where a agent learns the optimal policy for performing a sequential decision making without complete knowledge of the environment. Recent successes in deep reinforcement learning have enabled RL agents to solve complex problems from balancing inverted pendulums [10] to playing Atari games [8]. Despite these recent successes, we do not yet understand how to effectively transfer the learned policies from one task to another [11]. While some level of success has been obtained in transferring RL policies particularly in same domains tasks, the problem of efficient cross domain skill transfer is still quite open.

A major body of literature in Transfer Learning (TL) in the RL setting are focused on using the learned source policy as an initial policy in the target task [12, 2, 3]. Examples include transfer in scenarios where source and target task are similar and no mapping of state space is needed [4] or transfer from human demonstrations [9]. However, when source and target have different state-action spaces, the policy from source cannot be directly used in the target task. In case of cross-domain transfer a mapping is required between the state-action space of corresponding source and target tasks to enable knowledge transfer [11]. This inter task mapping can be supervised, provided by an agent [13], hand coded [7] or unsupervised using Manifold Alignment [3] or Sparse Coding Algorithm [2]. However TL with inter task mapping may accelerate learning and minimize regret in target task, but do not lead to a sample efficiency in transfer. In particular, these approaches do not leverage the fact that both tasks exploit the same physical principle and hence possibility of reusing source policy in target domain.

2 Main Contribution: Transfer Learning with Target Apprentice

To address this gap, our main contribution is an algorithm that enables cross domain transfer in the RL setting. The presented architecture is applicable to both continues and discrete state space systems with discrete action set. Unlike the available state of art TL algorithms, which mainly concentrates on policy initialization in target task RL; we propose to use source policy directly as the optimal policy in related target task. Leveraging notions of apprenticeship learning [1] and adaptive control [6], we achieve this one step transfer, through on-line correction of transfered policy by an adaptive policy derived based on model transition error. The presented approach has three distinct phases: Phase I involves Reinforcement learning in the source task to find the optimal policy. For this purpose, we use Fitted Value Iteration Q-Learning (FQI) [5] to solve the source task MDP. Phase II, using traces (s, a, s') state-action-state triplets collected executing random policy $\pi_r(s, a)$ in both the source and target tasks, an interstate mapping $\chi_{\mathbf{S}}$ is discovered. This mapping can be found using unsupervised manifold alignment [3] or sparse coding algorithm [2]. The inter task mapping enables mutual correspondence between source and target state spaces for transferring the learning. Phase III of transfer is adaptive policy generation and augmenting the transfered source policy. We do not use the transfered policy as initialization for target task RL, but rather adapt the policy so that it behaves as the optimal policy for the target task. Inspired by the literature in model reference adaptive control [6], we achieve this by adapting the policy to accommodate for the difference between the transition models of the projected source and the target task. The details of proposed TL architecture is presented in following subsection,

2.1 Transfer Learning through policy adaptation

This section presents a transfer algorithm for pair of tasks in continues/discrete state and discrete action spaces. Transfer learning through policy adaptation using apprentice model is described in Algorithm-1. We show empirically this method is sample efficient compared to other methods of transfer; since the sample complexity of RL on a initialized target task is reduced to sample complexity of local apprentice model learning around the optimal trajectory.

Algorithm-1 leverages the intertask mapping [3] to move back and forth between source and target space for knowledge transfer and adaptive policy learning. Performance of the policy transfer, depends on goodness of manifold alignment between source and target tasks. Algorithm-1 provides pseudo-code for transfer architecture and can be distinguished into two parts (1) Transfer (steps $1-8$) and (2) Adaptive policy learning (step $9-12$).

For every initial condition in target task $s_0^{(T)} \in \mathcal{S}^{(T)}$; $s_0^{(T)}$ are mapped using the inverse mapping $\chi_{\mathbf{S}}{}^{+}$, to find the corresponding initial condition on state space in source task (step-4), where $\hat{s}_i^{(S)}$ represents the image of $s_i^{(T)}$ in source space. For the mapped state in source task a greedy action is selected using learned $Q^{(S)}(\hat{s}_i^{(S)}, a_i)$ action value function (step-5 & 6).

Assumption:1 We assume the cardinality of the discrete action space is same in source and target task, $|\mathcal{A}^{(T)}| = |\mathcal{A}^{(S)}|$

Using selected action $a_i^{(S)}$, the source model at state $\hat{s}_i^{(S)}$ can be propagated to $\hat{s}_{i+1}^{(S)}$ (step-7), we assume either transition model for the source task is available or that we can sample from a source model simulator. This assumption is not very restrictive, since designer can always select/create related source task for given target task. The propagated state

Algorithm 1 Transfer Learning using target Apprentice model

```
1: Input: Source Policy $\pi^{*(S)}(s,a)$, Inter-task mapping $\chi_{\mathcal{S}}$ and Apprentice Model $\hat{\mathcal{P}}^{(T)}$
2: repeat
3:     Initialize $(s_0, a_0)^{(T)} \in (\mathcal{S} \times \mathcal{A})^{(T)}$.
4:     Project the target task state to source model using inter-task mapping ${\chi_{\mathcal{S}}}^{+}$
           $\hat{s}_i^{(S)} = {\chi_{\mathcal{S}}}^{+}(s_i^{(T)})$
5:     Evaluate the action using greedy policy on learned action-value function $Q^{(S)}(s,a)$ in source task,
           $\pi_i^{*(S)} = arg\max_{\pi}(Q^{(S)}(\hat{s}_i^{(S)}, a))$
6:         $a_i^{(S)} = \pi^{*(S)}(\hat{s}_i^{(S)}, a_i)$
7:     Query the source task model at state $\hat{s}_i^{(S)}$ and action $a_i^{(S)}$
           $s_{i+1}^{(S)} = \mathcal{P}^{(S)}(\hat{s}_i^{(S)}, a_i^{(S)})$
8:     Project the source task propagated state to target task model,
           $\hat{s}_{i+1}^{(T)} = \chi_{\mathcal{S}}(s_{i+1}^{(S)})$
9:     Evaluate the adaptive policy as
           $\pi_{ad}^{(T)} = \hat{\mathcal{P}}^{(T)}(s_i^{(T)}, a_i) - \mathcal{P}_{(S)}^{(T)}(s_i^{(T)}, a_i)$
10:    Project the source policy into target space $\chi_s(\pi^{*(S)}(s,a))$
11:    TL policy for target task
           $\pi^{TL} = \chi_{\mathbf{S}}(\pi^{*(S)})(s,a) - \pi_{ad}^{(T)}(s,a)$
12:    Draw action from policy $\pi^{TL}$ in TL setting at step $i$, and propagate the target model
           $a_i^{TL} = a_{(S)i}^{(T)} - a_{ad,i}^{(T)}$
13: until $s_i^{(T)} \neq terminal$
```

in source task can be mapped back to target space using inter-task mapping function (step-8), where $\hat{s}_{i+1}^{(T)}$ is the image of $s_{i+1}^{(S)}$, in target space. From Assumption-1, every selected action in source task has greedy correspondence in target task. Using this equivalence of actions, for every selected action in source task an equivalent action in target task is selected as $a_i^{(T)} \in \mathcal{A}^{(T)}$ (step-10). We augment the selected action for target task with adaptive action $a_{ad}^{(T)} \in \mathcal{A}_{ad}^{(T)}$ derived from adaptive policy (step-11), where $\hat{\mathcal{P}}^{(T)}(s_i, a_i) = A^{(T)}\phi(s_t) + Bu_t$ is apprentice model, $\mathcal{P}_{(S)}^{(T)}(s_i, a_i) = \chi_{\mathcal{S}}(\mathcal{P}^{(S)}(s_i, a_i))$ is projected source model on to target space and $\mathcal{A}_{ad}^{(T)}$ is adaptive action space such that $|\mathcal{A}_{ad}^{(T)}| \geq |\mathcal{A}^{(T)}|$ and $||\mathcal{A}_{ad}^{(T)}|| \leq ||\pi_{ad}^{(T)}||_{\infty}$. Therefore the total transfered policy in target space in solving a related target task is proposed to be a linear combination of mapped optimal policy and adaptive policy as follows,

$$\pi^{*(T)} = \chi_{\mathcal{S}}(\pi^{*(S)}) - \pi_{ad}^{(T)}$$

3 Experiments & Results

We present results from three experiments to evaluate the proposed transfer learning framework. The first two experiments consider transfer learning in domains with the same state-space but with different transition models and action spaces. To demonstrate the quality of transfer, we compare the proposed Target Apprentice TL (TA-TL) against existing transfer learning framework via manifold alignment (MA-TL) [3] and stand alone RL on target domain with no initialization from source task learning.

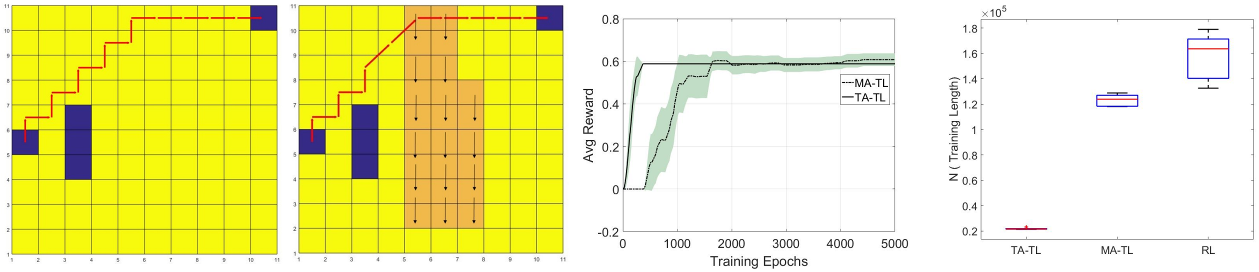
We learn the optimal policy in the source task using FQI with shallow, single layer Radial Basis Functions (RBF) network. For similar domain transfer, the state space for both source and target are same but for cross domain problem, the state space differ in their span and size. We also do not need target reward model be similar to the source task, as the proposed algorithm directly adapts the Q function from the source task and does not engage in RL in the target domain.

3.1 Grid World to Windy Grid World

The source task in this problem is Non-Windy (NW) grid world. The state variables describing the system are grid positions. The RL objective is to navigate an agent through obstacles from start to goal position optimally. The admissible actions are up $[0, +1]$, down $[0, -1]$, right $[+1, 0]$ and left $[-1, 0]$. The reward function is $(+10)$ for reaching goal position, (-1) for hitting obstacles and (0) every where else. The target domain is same as the source but with added wind bias which affects the transition model in parts of the state-space (see Figure 1b).

The optimal policy in source task (non-windy grid world) $\pi^{*(S)}$ is learned using FQI. Since the source and target state space are identical, inter-task mapping is not needed. We start with 100 randomly sampled starting position and execute policy $\pi^{*(S)}$ in target domain and collect samples for apprentice model learning. Empirically, we show the proposed method (TA-TL) provides a sample efficient TL algorithm compared to other transfer techniques.

Figure 1a and 1b shows the results of same domain transfer in grid world, demonstrating TA-TL achieving successful transfer in navigating through grid with obstacles and wind. Figure 1c and 1d shows the quality of transfer through faster convergence to average maximum reward with lesser training samples compared to MA-TL and RL methods. The presented algorithm is able to attain maximum average reward in reaching goal position in $\sim 2 \times 10^4$ steps. MA-TL and RL algorithm achieve similar performance in $\sim 1.2 \times 10^5$ and $\sim 1.7 \times 10^5$ steps respectively, nearly one order higher compared to proposed TA-TL.

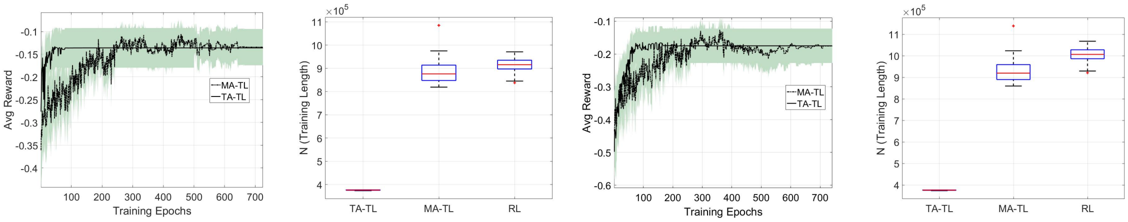


(a) (b) (c) (d)

Figure 1: Non windy to Windy grid World Transfer:(a) & (b) Agent navigating through grid world in source and target domain (c) Average Rewards & (d) Training length, Comparing quality of transfer for TA-TL and MA-TL through convergence rate of Average Reward and Training Length

3.2 Inverted Pendulum (IP) to time-varying IP

We demonstrate our approach for a continuous state domain Inverted Pendulum (IP) swing-up and balance. The source task is the conventional IP domain [10]. The target task is a non-stationary inverted pendulum where the length and mass of the pendulum are continuously time varying with function $L_i = L_0 + 0.5cos(\frac{\pi i}{50})$ and $M_i = M_0 + 0.5cos(\frac{\pi i}{50})$, where $L_0 = 1$, $M_0 = 1$ and $i = 1 \dots N$. The state variables describing the system are angle and angular velocity $\{\theta, \dot{\theta}\}$ where $\theta, \dot{\theta} \in [-\pi, \pi]$. The RL objective is to swing-up and balance the pendulum upright such that $\theta = 0, \dot{\theta} = 0$. The reward function is selected as $r(\theta, \dot{\theta}) = -10|\theta|^2 - 5|\dot{\theta}|^2$, which yields maximum value at upright position and minimum at the down-most position. The action space is: full throttle right $(+\tau)$, full throttle left $(-\tau)$ and zero throttle. Note that the domain is tricky, since full throttle action is assumed to not generate enough torque to be able to lift the pendulum to the upright position, hence the agent must learn to swing the pendulum so that it generates oscillations and leverages angular momentum to go to the upright position. The target task differs from the source task in the transition model.



(a) (b) (c) (d)

Figure 2: IP to Non-stationary IP Transfer: (a) Average Rewards and (b) Training length, MC to IP Transfer: (c) Average Rewards and (d) Training length

Figure 2a and 2b shows the quality of transfer through faster convergence to average maximum reward with lesser training samples compared to MA-TL and RL methods.

3.3 Mountain Car (MC) to Inverted Pendulum (IP)

We have tested the cross domain transfer between mountain car to inverted pendulum. The source and target task are characterized by different state and action space. The source task, MC is a benchmark RL problem of driving an underpowered car up a hill. The dynamics of MC are described by two continues state variables $(x, \dot{x})$ where $x \in [-1.2, 0.6]$

and $\dot{x} \in [-0.07, 0.07]$. The input action takes three distinct values $(+1)$ full throttle right, (-1) full throttle left and (0) no throttle. The reward function is proportional to negative of the square distance of car from goal position. The target task is conventional IP as described in previous experiment. Cross domain transfer requires a correspondence between inter-task space manifold for mapping the learned policy and source transition model from source to target space and back. We use unsupervised manifold alignment [3] technique to obtain this mapping. We do not report the training time to learn the intertask mapping, since it is common to both TA-TL and MA-TL methods. We used a random policy to generate samples for manifold alignment and projected source policy in target space for target apprentice learning [1].

For all above results the training length involved with TA-TL method in Figure 2d, 2b, and 1d is sample lengths for target apprentice learning. We compare TA-TL with MA-TL and generic RL on target task. We compare the efficiency and effectiveness of transfer methods based on sample efficiency in learning the new task and rate of convergence to maximum average reward. Similar to the same domain transfer Figure 2c and 2d shows the quality of transfer for TA-TL through faster convergence to average maximum reward with lesser training samples compared to MA-TL and RL methods.

4 Conclusions

We introduced a new Transfer Learning technique in RL, that leads to sample efficient transfer between source and target tasks. The presented approach demonstrates the near optimality of transfered policy in target domain through augmenting with a adaptive policy; which accounts for the model error between target and projected source. The sample complexity in transfer is reduced to target apprentice learning, which we demonstrated empirically, leads to more than one order improvement in training lengths over existing approaches for transfer learning in RL for both cross and same domain tasks.

References

[1] Pieter Abbeel and Andrew Y Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 1–8. ACM, 2005.

[2] Haitham B Ammar, Karl Tuyls, Matthew E Taylor, Kurt Driessens, and Gerhard Weiss. Reinforcement learning transfer via sparse coding. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 383–390. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[3] Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew E Taylor. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. In *Proc. of AAAI*, 2015.

[4] Bikramjit Banerjee and Peter Stone. General game learning using knowledge transfer. In *IJCAI*, pages 672–677, 2007.

[5] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*, volume 39. CRC press, 2010.

[6] Girish Chowdhary, Tongbin Wu, Mark Cutler, and Jonathan P How. Rapid transfer of controllers between uavs using learning-based adaptive control. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5409–5416. IEEE, 2013.

[7] George Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pages 895–900, 2007.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[9] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2219–2225. IEEE, 2006.

[10] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[11] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[12] Matthew E Taylor, Peter Stone, and Yaxin Liu. Value functions for rl-based behavior transfer: A comparative study. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 880. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[13] Lisa Torrey, Jude Shavlik, Trevor Walker, and Richard Maclin. Relational macros for transfer in reinforcement learning. In *International Conference on Inductive Logic Programming*, pages 254–268. Springer, 2007.

Generalized Inverse Reinforcement Learning

James MacGlashan[*][†]
Cogitai, Inc.
james@cogitai.com

Nakul Gopalan[*]
Brown University
ngopalan@cs.brown.edu

Michael L. Littman
Brown University
mlittman@cs.brown.edu

Amy Greenwald
Brown University
amy@cs.brown.edu

Abstract

Inverse Reinforcement Learning (IRL) is used to teach behaviors to agents, by having them learn a reward function from example trajectories. The underlying assumption is usually that these trajectories represent optimal behavior. However, it is not always possible for a user to provide examples of optimal trajectories. This problem has been tackled previously by labeling trajectories with a score that indicates good and bad behaviors. In this work, we formalize the IRL problem in a generalized framework that allows for learning from failed demonstrations. In our framework, users can score entire trajectories as well as individual state-action pairs. This allows the agent to learn preferred behaviors from a relatively small number of trajectories. We expect this framework to be especially useful in robotics domains, where the user can collect fewer trajectories at the cost of labeling bad state-action pairs, which might be easier than maneuvering a robot to collect additional (entire) trajectories.

Keywords: Inverse Reinforcement Learning

---

[1] Both authors contributed equally to this work.
[2] Work was done while at Brown University.

1 Introduction

Teaching desired behaviors to an agent is an important problem in artificial intelligence. One approach to this problem is to use Inverse Reinforcement Learning (IRL) [1, 4], in which demonstration trajectories are provided, based on which behavior is learned, by first learning a reward function, which is an intermediate generalizable representation of behavior. Recent IRL methods have considered scoring the trajectories collected, so that the agent learns to prefer behaviors with high scores and avoid behaviors with low scores. Our work builds on this idea, but goes one step further by allowing the user to point out the reasons for a high or a low score. More formally, we propose a generalized framework for IRL, where it is possible to score the state-actions pairs of a trajectory, as well as the trajectory itself. The hope is that this added information might facilitate the training of complex behaviors from fewer demonstration trajectories in problem domains like robotics, where it is particularly expensive to collect trajectories.

In this short paper, we first summarize the state of the art in Inverse Reinforcement Learning. We then discuss our framework, as well as two learning algorithms that can perform inference over the proposed framework. Finally, we show example reward functions and behaviors learned from demonstrations using our algorithms for a sample grid world problem.

2 Related Work

Two competing approaches to teaching behaviors to agents are supervised learning[6] and reinforcement learning [1, 4]. The advantage of using Reinforcement learning over supervised learning is that it allows the agent to repeat the learned behavior in a different environment than the one it was trained in, as a reward function is learned. A reinforcement learning problem is generally modeled as a Markov Decision Process (MDP), which is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \mathcal{R}, \gamma)$, as defined in [8]. In some applications, it can be very difficult to design a reward function $\mathcal{R}$, but it might be easy to provide example trajectories as behaviors to learn from. Inverse Reinforcement Learning [1, 4] is a problem reformulation, where behaviors are provided, and the goal is to teach agents behaviors by having them learn reward functions from example trajectories. Traditional Inverse Reinforcement learning approaches did not let agents learn via failure; instead, demonstrations were interpreted as optimal behaviors.

Previous methods [3, 7] for learning from imperfect demonstrations scored the trajectories. Burchfiel et al. [3] tries to estimate the weights of a parameterized reward function by learning optimal weights for the scoring function, given trajectories and scores from experts and non-experts. Shiarlis et al. [7] assumes a labeled dataset with failed and successful trajectories, and learns the weights for a parameterized reward function such that the expected policy under the learned reward function is far away from the failed demonstrations and close to the successful demonstrations. Both works report that they require fewer trajectories than vanilla IRL to learn complicated behaviors, as more label information is available. We propose providing still more direct information, as a user might be able to label bad segments of a trajectory directly. This approach would be useful in scenarios where collecting data is harder than labeling segments of a trajectory as good or bad.

3 Generalized IRL (GIRL)

In this section, we develop a graphical model for the IRL problem over which inference can be performed to solve for a parameterized reward function. We then describe inference techniques that can be used within this framework to solve for the reward function.

3.1 Graphical Model

Consider a demonstration trajectory of length N given to the user and the agent. The user might provide a final label L to the trajectory demonstrated. This label L is considered by us to be a random function of what they thought, positive or negative, about each of the actions A selected by the agent in the trajectory. The motivation for this model is that we consider a final label (L) that a user gives a trajectory of size N to be some random function of what they thought about each of the action selections (A) exhibited in the trajectory. However, these step-wise evaluations (X) are mostly unobserved in the data, unless specifically labeled by the user. The reward function parameterized by $\boldsymbol{\theta}$ that dictates the stepwise evaluation is also an unobserved variable. We have shown the plate model in Figure 1, where the observed variables are in shaded nodes and the unobserved variables are in unshaded nodes.

We model the probability that an action is evaluated as good or not as proportional to its selection probability according a softmax policy computed for the reward function with parameters $\boldsymbol{\theta}$. Specifically:

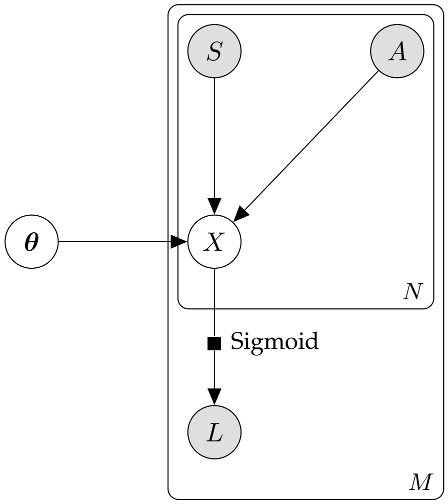


Figure 1: Plate Diagram of the Label Probability Model

$$\Pr(x_i = +1|s, a, \theta) = \pi(s, a|\boldsymbol{\theta}) \tag{1}$$
$$\Pr(x_i = -1|s, a, \theta) = 1 - \pi(s, a|\boldsymbol{\theta}), \tag{2}$$

where $\pi(s, a|\boldsymbol{\theta})$ is the softmax policy over Q-values computed for the reward function parameterized by $\boldsymbol{\theta}$:

$$\pi(s, a|\boldsymbol{\theta}) = \frac{e^{\beta Q(s,a|\boldsymbol{\theta})}}{\sum_{a'} e^{\beta Q(s,a'|\boldsymbol{\theta})}}, \tag{3}$$

β is a selectable parameter, and $Q(s, a|\boldsymbol{\theta})$ is the Q-value computed for the reward function parameterized by $\boldsymbol{\theta}$. This idea of having state-action pair labels have a probability distribution with respect to the expected optimal policy is similar to the ideas explored in SABL [5].

For the probability distribution of L, given the sequence of N step-wise labels, we would like a distribution that has the property that as more step-wise labels are positive, the probability of a positive trajectory label increases (and vice versa). Although there are many possible distributions that satisfy this property, for concreteness, we choose the sigmoid function. That is,

$$\Pr(L = +1|X_1, ..., X_n) = \frac{1}{1 + e^{-\phi \sum_i^N X_i}} \tag{4}$$
$$\Pr(L = -1|X_1, ..., X_n) = 1 - \Pr(L = +1|X_1, ..., X_n), \tag{5}$$

where ϕ is a selectable parameter that tunes how quickly of a majority of step-wise labels increases/decreases the probability of trajectory assignment. For example, when $\phi = 0$, trajectory labels are assigned uniformly randomly independently of step-wise labels. As $\phi \to \infty$, the sigmoid converges to a step function in which a trajectory containing even one more positive step-wise label than negative step-wise label will deterministically cause a positive trajectory label (and vice versa).

The label probability model described above gives a generalized IRL formulation over which inference can be performed, with or without labels. If the given data set has no trajectory labels, then all trajectories can be set to have the same positive label. Next we will describe inference over this model using Expectation Maximization (EM) and a faster EM method using importance sampling.

3.2 Inference

The problem with estimating the $\boldsymbol{\theta}$ parameters of our reward function is that we have a latent variable vector X (or more generally, some of the elements of the X vector are latent, and some may be observed), which prevents us from easily computing the likelihood of the model and maximizing parameters for it. The EM approach to solving this problem is to first choose values for $\boldsymbol{\theta}$; then choose a new $\boldsymbol{\theta}$ that maximizes the expected value of the log likelihood function where the distribution of the expectation is the probability distribution of latent variables (the Xs in our case) given the observations available and previous $\boldsymbol{\theta}$ choice; and then repeating this process. The maximization process can be performed using gradient ascent, which is similar to the ideas explore in MLIRL [2].

To formalize this process for our problem, first note that the likelihood of our parameters (and state-action sequence) given an $\boldsymbol{x}$ vector and label l is

$$\mathcal{L}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}|l, \boldsymbol{x}) = \Pr(l|\boldsymbol{x}) \prod_i \Pr(x_i|s_i, a_i, \boldsymbol{\theta}) \tag{6}$$

and the log likelihood is

$$\log \mathcal{L}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}|l, \boldsymbol{x}) = \log \Pr(l|\boldsymbol{x}) + \sum_i \log \Pr(x_i|s_i, a_i, \boldsymbol{\theta}). \tag{7}$$

Additionally, the gradient of the log likelihood is,

$$\nabla_{\boldsymbol{\theta}} \log \mathcal{L}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}|l, \boldsymbol{x}) = \sum_i \frac{\nabla_{\boldsymbol{\theta}} \Pr(x_i|s_i, a_i, \boldsymbol{\theta})}{\Pr(x_i|s_i, a_i, \boldsymbol{\theta})}. \tag{8}$$

To simplify the EM algorithm description, we will introduce the notation $\boldsymbol{x}_k$ to indicate the subset of observed elements in an $\boldsymbol{x}$ vector, and $\boldsymbol{x}_u$ to represent a possible assignment to the subset of the unobserved values of an $\boldsymbol{x}$ vector. Using this notation, the expected value of the log likelihood under some candidate parameter $\boldsymbol{\theta}'$ for missing X elements distributed according to $\boldsymbol{\theta}$ is

$$E_{\boldsymbol{x}_u \sim \Pr(\boldsymbol{x}_u|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}\left[\log \mathcal{L}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}'|l,\boldsymbol{x})\right] = \sum_{\boldsymbol{x}_u} \Pr(\boldsymbol{x}_u|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}) \log \mathcal{L}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}'|l,\boldsymbol{x})$$

Given that, the corresponding EM algorithm operating on a single trajectory is as follows (it is generalized to many trajectories by simply summing over each trajectory).

Algorithm 1 Labeled-IRL EM Algorithm

```
Require: initial $\boldsymbol{\theta}_0$, and data $\boldsymbol{s}$, $\boldsymbol{a}$, $\boldsymbol{x}_k$, $l$
  for $t = 0$ to $K$ do
    $\boldsymbol{\theta}_{t+1} \leftarrow \arg\max_{\boldsymbol{\theta}'} \sum_{\boldsymbol{x}_u} \Pr(\boldsymbol{x}_u|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}_t) \log \mathcal{L}(\boldsymbol{\theta}',\boldsymbol{s},\boldsymbol{a} \mid l,\boldsymbol{x}_k,\boldsymbol{x}_u)$
  end for
```

To compute the expected value, we need to enumerate each of the possible assignments to the unknown $\boldsymbol{x}$ elements and compute the probability of them given the observed data and model parameters $\boldsymbol{\theta}$. This probability is computed as

$$\begin{aligned}\Pr(\boldsymbol{x}_u|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}) &= \frac{\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u)\Pr(\boldsymbol{x}_u|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})\Pr(\boldsymbol{x}_k|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})\Pr(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}{\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})\Pr(\boldsymbol{x}_k|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})\Pr(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} \\ &= \frac{\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u)\Pr(\boldsymbol{x}_u|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}{\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} \\ &= \frac{\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u)\prod_i \Pr(\boldsymbol{x}_{u,i}|s_i,a_i,\boldsymbol{\theta})}{\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}.\end{aligned}$$

A straightforward computation of $\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$ requires marginalizing over all possible assignments to the unknown X elements; however we can exploit the fact that $\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$ is a function of the sum of the feedback values, the marginalization can be reduced to a summation that iterates over a number of values that is a linear function of the number of unobserved feedbacks.

Unfortunately, even with an efficient means to compute $\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$, when the number of unknown X variables is large, the number of $\boldsymbol{x}_u$ assignments enumerated in the expectation's outer sum grows exponentially, and the product series over each of unknown element probabilities in the above equation ($\prod_i \Pr(\boldsymbol{x}_{u,i}|s_i,a_i,\boldsymbol{\theta})$) can have underflow issues. A resolution to this problem is to estimate the expectation with sampling. Monte Carlo sampling is unfortunately intractable because it is not easy to sample from $\Pr(\boldsymbol{x}_u|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$; moreover, it would not address the underflow issue in the product series. However, it is easy to sample from $\Pr(\boldsymbol{x}_u|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$ (removing the conditioning on the label), which we can use in importance sampling. With importance sampling, we can replace the expectation computation with the sample-estimate

$$\frac{1}{C}\sum_{j}^{C} \frac{\Pr(\boldsymbol{x}_u^j|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}{\Pr(\boldsymbol{x}_u^j|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} \log \mathcal{L}(l,\boldsymbol{x}_k,\boldsymbol{x}_u^j|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta}), \tag{9}$$

where $\boldsymbol{x}_u^j$ is sample from the distribution $\Pr(\boldsymbol{x}_u|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$. This simplifies further to:

$$\begin{aligned}\frac{\Pr(\boldsymbol{x}_u^j|l,\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}{\Pr(\boldsymbol{x}_u^j|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} &= \frac{\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u)\Pr(\boldsymbol{x}_u^j|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}{\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}\frac{1}{\Pr(\boldsymbol{x}_u^j|\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})} \\ &= \frac{\Pr(l|\boldsymbol{x}_k,\boldsymbol{x}_u)}{\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})}\end{aligned}$$

Consequently, we have removed the product series from the expectation weight, thereby avoiding underflow issues. Also, as noted previously, the $\Pr(l|\boldsymbol{x}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{\theta})$ term can be computed efficiently with dynamic programming. Now we can write a tractable EM algorithm where we can compute the maximization using gradient ascent.

Algorithm 2 Labeled-IRL Approximate EM Gradient Ascent Algorithm

```
Require: initial $\boldsymbol{\theta}_0$; data $\boldsymbol{s}$, $\boldsymbol{a}$, $\boldsymbol{x}_k$, $l$; and learning rate $\alpha$
  for $t = 0$ to $K$ do
      draw $j = 1$ to $C$ samples of $\boldsymbol{x}_u^j \sim \Pr(\boldsymbol{x}_u | \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}_t)$
      for $j = 1$ to $C$ do
          $w_j \leftarrow \frac{\Pr(l|\boldsymbol{x}_k, \boldsymbol{x}_u)}{\Pr(l|\boldsymbol{x}_k, \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}_t)}$        ▷ Expectation step
      end for
      $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}_t$
      for 1 to $M$ do        ▷ Gradient ascent maximization loop
          $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}' + \alpha \frac{1}{C} \sum_j^C w_j \sum_{x_i \in \boldsymbol{x}_k \cup \boldsymbol{x}_u^j} \frac{\nabla_{\boldsymbol{\theta}'} \Pr(x_i | s_i, a_i, \boldsymbol{\theta}')}{\Pr(x_i | s_i, a_i, \boldsymbol{\theta}')}$
      end for
      $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}'$
  end for
```

4 Results on Grid World

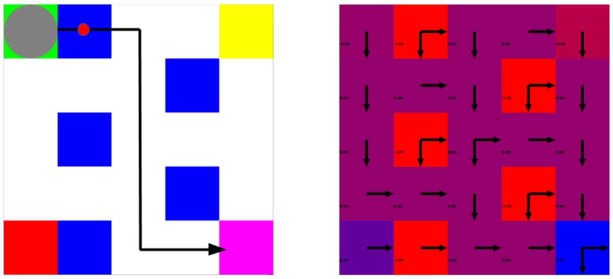

(a) (b)

Figure 2: (a) Shows the single input trajectory to train the behavior. The red dot shows the state for which the agent got a negative feedback. (b) Shows the output policy and Value function learned for each state, given the labeled trajectory.

We present results on a grid world, using the EM method. The grid world is shown in Figure 2a. We want to teach a behavior in which the agent walks to the pink cell, while avoiding the blue cells. During demonstration the agent walks over a single blue cell. We mark the state-action pair that landed the agent in the blue with a negative feedback, shown with a red dot, in Figure 2a. We used one trajectory with eight steps, with $\beta = 1$, and $\phi = 1$. The overall label for the trajectory is positive as the agent reached the pink cell. After performing GIRL we learn a reward function and we have displayed the value function and the optimal policy for the entire grid as shown in Figure 2b. It is obvious that the agent now prefers to walk to the pink grid cell while avoiding the blue cells. Teaching this behavior would require two separate trajectories, if failed trajectories were included, in all the state of the art failure IRL methods discussed previously. Hence, we can learn complex behaviors with fewer example trajectories.

5 Conclusion

Our approach allows us to provide individual state-action pair feedback for IRL, along with labels on state-action pairs, which allows for more expressive feedback with fewer sample trajectories. The inference algorithms we developed performs this inference in real time on our toy example. The algorithm itself will be very useful in teaching behaviors with active learning as well as in two agent games, where one agent teaches the rules of the game to the other.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[2] Monica Babes, Vukosi Marivate, Kaushik Subramanian, and Michael L Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 897–904, 2011.

[3] Benjamin Burchfiel, Carlo Tomasi, and Ronald Parr. Distance minimization for reward learning from scored trajectories. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[4] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *ICML*, pages 663–670, 2000.

[5] Bei Peng, James MacGlashan, Robert Loftin, Michael L Littman, David L Roberts, and Matthew E Taylor. A need for speed: adapting agent action speed to improve task learning from non-expert humans. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 957–965, 2016.

[6] Stefan Schaal et al. Learning from demonstration. *Advances in neural information processing systems*, pages 1040–1046, 1997.

[7] Kyriacos Shiarlis, Joao Messias, and Shimon Whiteson. Inverse reinforcement learning from failure. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1060–1068, 2016.

[8] Philip S. Thomas. A notation for markov decision processes. *CoRR*, abs/1512.09075, 2015. URL `http://arxiv.org/abs/1512.09075`.

Adversarial Attacks on Deep Reinforcement Learning

Anay Pattanaik
Coordinated Science Lab
Department of Agriculture and Biological Engineering
University of Illinois at Urbana-Champaign
`anayp2@illinois.edu`

Victor Sui
Department of Computer Science
University of Illinois at Urbana-Champaign
`vsui2@illinois.edu`

Denis Osipychev
Coordinated Science Lab
Department of Agriculture and Biological Engineering
University of Illinois at Urbana-Champaign
`deniso2@illinois.edu`

Girish Chowdhary [*]
Department of Agriculture and Biological Engineering
Coordinated Science Lab
University of Illinois at Urbana-Champaign
`girishc@illinois.edu`

Abstract

This paper engineers attacks on reinforcement learning (RL) algorithms. The RL policies learnt with deep neural network (DNN) and radial basis function network (RBF) as underlying function approximators are compared against each other from the perspective of robustness to adversarial attack. Learnt policies are attacked by inducing adversarial noise in observation for the algorithm during executing phase. Interesting, we show that a naively engineered adversarial attack successfully degrades the performance of deep reinforcement learning whereas there is no significant degradation of equivalent radial basis network based reinforcement learning policy. We provide results synchronous to adversarial attacks on classification of images where RBF were more robust to attacks since they could not generate any label with confidence as opposed to confidently generating wrong labels from adversarial attacks as done by DNN.

Keywords: Deep Learning, Reinforcement Learning, Deep Reinforcement Learning, Adversarial Examples

---

[*] Asst Professor, Agricultural and Biological Engineering, UIUC (http://daslab.illinois.edu/drupal/home)

1 Introduction

Reinforcement learning in general has achieved several success in computer vision [5], dialog generation [7] and in robotics domain [6], [10]. In robotics, it offers an interesting alternative to control systems which is primarily based on domain specific models of the plants. Thus, RL algorithms in principle can provide us with a controllers that are not application dependent. However, this comes at the cost of curse of dimensionality wherein it is very difficult to scale naive RL algorithms in real life. In those situations, RL algorithms require feature engineering and function approximators to take care of the curse of dimensionality. Speaking of these, DNN has a tremendous impact in the primarily in the field of superviesd learning [5] with no requirement of hand crafted features. Thus, using DNN as function approximator in RL algorithm has been proposed [8], [9] and offers state-of the art results in several RL tasks [11], [9].

However, Goodfellow et al. [2] have shown that Deep Networks can be fooled easily into predicting wrong label by perturbing the input with adversarial attacks. It opens up interesting frontier regarding adversarial attacks on machine learning algorithms in general. This assumes greater importance in the context of robotics and safety critical systems where such adversarial noise may lead to undesirable and hazardous situations. An interesting avenue is to see whether we can enginner such attacks on reinforcement learning algorithms which is the basis of this paper.

The contribution of this paper is to engineer adverserial attacks on reinforcement learning algorithms in general by injecting small noise in the observation space of the algorithm. We also examine robustness to such attacks for different function approximators namely deep neural network and radial basis function network. We have shown that for different RL algorithms, RBF is significantly more robust than DNN for these naively generated attacks.

1.1 Related Work

Very recently, Huang et al. [4] have presented interesting results regarding adversarial attack on reinforcement learning algorithms. However, those attacks are similar in spirit to fast signed gradient descent (FSGD) in [2]. In [4], instead of probability of the image being classified with a label is replaced by the probability of taking an action as in [2]. Thus, they have generated results using RL algorithms for Atari game environment (using image observation). This work provides complimentary results. We have focussed on engineering attacks for RL algorithms in general with any physical state as observation (not restricted to images and discrete action space). Thus, our work caters to reinforcement learning in general which use value function aproximation in some form such as Deep Q learning (DQN) and Deep Deterministic Policy Gradient (DDPG). This also enables us to compare with RBF based reinforcement learning algorithms in similar domain. Another major difference is the use of value function to determine whether our attack is successful. In [4], the adversarial noise is generated without being certain (when the noise is generated) that it indeed causes adversarial attack. Thus, we are exploiting the value function to ascertain the efficacy of our attack.

2 Background

2.1 Deep Q Learning (DQN)

The initial algorithm that arguably lead to surge in interest of Deep Reinforcement Learning is Deep Q learning with superhuman results results on Atari games [9]. Q learning is a value function based algorithm (we refer to both state-action value function and state-value function as value function). Q values of a state-action pair represent how good the action being contemplated by the agent is for its current observation. The learning agent updates the Q value using temporal difference error and simultaneously acts to maximize its return in the long run. In deep Q learning algorithm, the agent uses a Deep neural network to approximate this Q function. The DQN algorithm achieved stability in training primarily by using experience replay and the use of target network. For experience replay, it stored past state, action, reward, next state sequence and used to to update the Q network just as in supervised learning with these sequences being picked randomly from memory. This breaks the strong correlation between samples. For this ”supervised” learning type of update, it had another neural network, called target network which provides the Q values for the next state(s). The target network is updated by hard transfer of online weights after a fixed number of iterations. This two network structure led to stability in training.

2.2 Radial Basis Function based Q learning

In radial basis function based Q learning [1], the DNN is replaced by RBF with gaussian kernel. Here experience replay and target networks are typically not used. The networks learns through stochastic gradient descent using temporal difference error.

2.3 Deep Deterministic Policy Gradient (DDPG)

Deep deterministic policy gradient (DDPG) [8] uses both an actor and a critic for learning. The critic evaluates (learns the value function) a given policy which is generated by the actor. The weights of critic and actor are updated by gradient descent. The critic network uses the prior concepts of experience replay and target network for its update. The only difference between these update and that of Deep Q learning is that instead of ”hard” transfer of weights to target after a fixed number of iterations, there is ”soft” transfer of weights where the weights of target network are incremented by a very small amount towards the online network. The actor network learns through gradient of the policy.

2.4 Radial Basis Function based Actor-Critic

This is similar to DDPG except that the actor and critic are represented by RBF [3]. Here, experience replay and target network are typically not used. The learning takes place through stochastic gradient descent for both actor and critic.

3 Method

We show that a naively engineered attack is good enough to generate adversarial attacks on Deep Reinforcement learning policy. The agent is first trained through several reinforcement learning algorithms and then attacked . An important point to note is that the attacks are generated during test phase. The adversarial attack is essentially a search across nearby observation which will cause the agent to take wrong action. For generating adversarial attack on the Deep Reinforcement Learning policies, we sample a noise with finite (small) support. The quality of noise is ascertained by value function approximation. Noise is not generated once during every iteration, rather it is generated for a number of times every iteration with a search for ”best” adversarial noise. The particular noise that causes least estimate of the value function is selected as adverserial noise. It is worth noting that a search for noise could have been performed by sampling value function gradient direction for a number of times instead of a single sample of step size epsilon, but it has not been pursued here because the random sampling from a distribution is good enough to attack the deep RL algorithm. As we shall show, this kind of attack is not enough to cause performance degradation in RBF based reinforcement learning algorithm. It might be noted that an attack based on gradient of value function might break RBF based RL policy which we will try in future.

4 Results

All the experiments have been performed within OpenAi gym environment. We performed the experiments using two reinforcement learning algorithms DQN and DDPG and their RBF counterparts. DQN has been applied on cart pole while DDPG has been applied to continuous mountain car problem.

4.1 Architecture for Deep Q Learning (DQN)

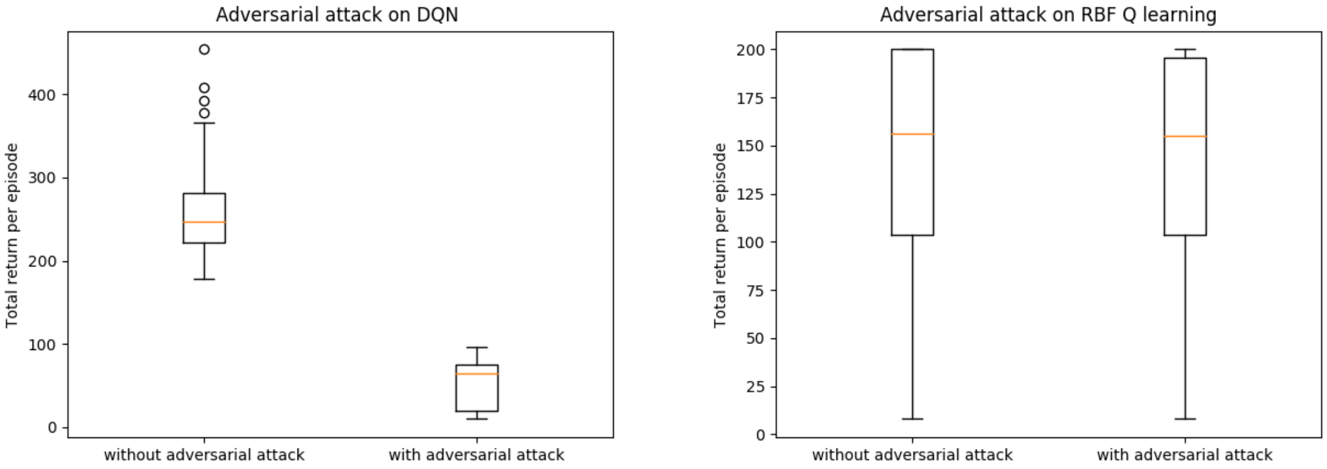
For deep Q learning, the Q network has 3 hidden layers each consisting of 16 ReLu units and two output units, each corresponding to force being applied in left and right direction. The corresponding adversarial noise are generated by mean shifted beta noise (with mean zero). The alpha nad beta parameters for the beta distribution were 0.3 and 0.3. The support for noise were restricted to 5% of the corresponding observation dimension.

4.2 Architecture for Radial Basis Function Based Q learning

For RBF based Q learning, the single hidden layer consisted of 64 hidden units and the output layer contained 2 units, each corresponding to force being applied to left and right direction. The centres were scattered uniformly across the entire state space with each dimension being divided in to a bin of size 3. The noise induced here were identical to the one for DQN.

4.3 Architeture for Deep Deterministic Policy Gradient (DDPG)

For DDPG, the actor network consisted of 3 layers of 32 hidden units and the output was action. The activation function for all the hidden units were ReLu. The exploration was determined by Ornstein-Uhlenbeck process with $\mu = 0$ and $\sigma = 0.3$ and $\theta = 0.5$. The critic layer consisted of the observation and action as input, 2 layers of 16 units with ReLu activation function were used as hidden layers. The adversarial noise that was induced here was uniform noise whose value of support was around 5% the values taken by observations in the corresponding state.

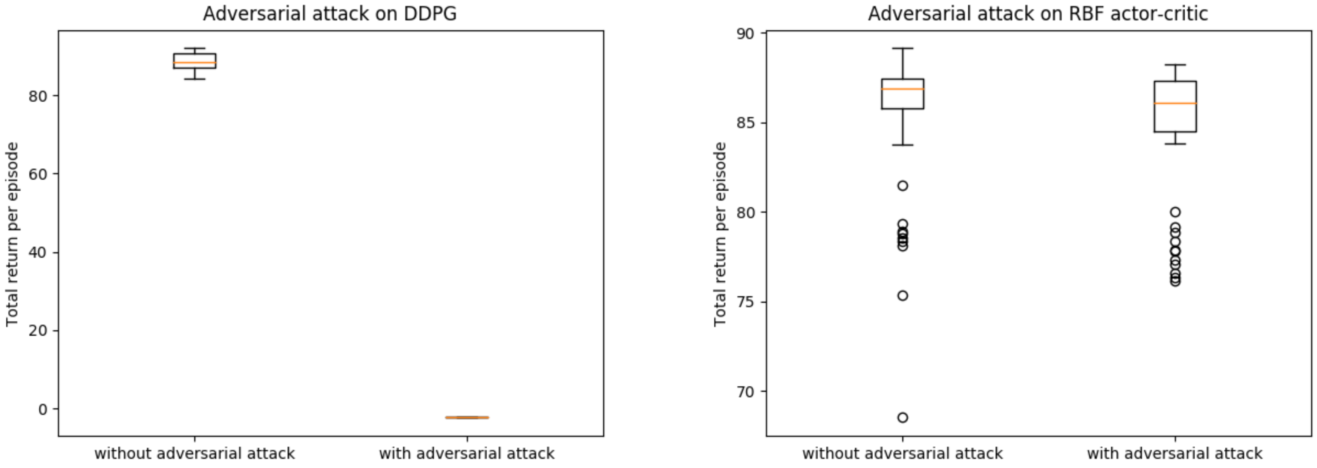


(a) Box plot comparing episode returns with and without attack (b) Box plot comparing episode returns with and without attack

Figure 1: Figures 1a and 1b show the boxplots of total reward obtained per episode for 100 episodes. It is clear that the naive adversarial attack has significantly degraded the performance of DQN while there is no significant degradation in performance of RBF based Q learning

4.4 Architecture for Radial Basis Function based Actor-Critic

For RBF based actor critic, the actor consisted of 64 nodes with Gaussian kernels. Again, the centres were scattered uniformly over the observation space. The critic network consisted of 400 nodes of Gaussian kernels with centres scattered uniformy over action space. The adversarial noise was identical to the one for DDPG.



(a) Box plot comparing episode returns with and without attack (b) Box plot comparing episode returns with and without attack

Figure 2: Figures 2a and 2b show the boxplots of total return obtained per episode for 100 episodes. It is clear that the naive adversarial attack has significantly degraded the performance of DDPG while there is no significant degradation in performance of RBF based actor-critic

5 Conclusion

In this paper, we have shown that reinforcement learning algorithms that have been trained with deep learning as function approximators can be fooled easily as compared to reinforcement learning algorithms that are based on radial basis function network. Interestingly, simple adversarial attacks on Deep RL can degrade it as opposed to robust policy learnt by RBF. Future work remains as to improve the attacks by using the gradient information of the value function and further looking into ways to mitigate them.

References

[1] Alborz Geramifard, Thomas J. Walsh, Stefanie Tellex, Girish Chowdhary, Nicholas Roy, and Jonathan P. How. A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Found. Trends Mach. Learn.*, 6(4):375–451, December 2013. ISSN 1935-8237. doi: 10.1561/2200000042. URL `http://dx.doi.org/10.1561/2200000042`.

[2] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[3] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

[4] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[7] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

[8] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[10] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

[11] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Learning Forest Wildfire Dynamics from Satellite Images Using Reinforcement Learning

Sriram Ganapathi Subramanian
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
`s2ganapa@uwaterloo.ca`

Mark Crowley
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
`mcrowley@uwaterloo.ca`

Abstract

Forest wildfires are a perennial problem in many parts of the world requiring high financial and social costs to measure, predict and control. One key challenge is modelling the dynamics of forest wildfire spread itself which usually relies on computationally expensive, hand crafted physics-based models. The question we ask is: Can we learn a dynamics model by treating wildfire as an agent spreading across a landscape in response to neighbourhood environmental and landscape parameters? The problem is modelled as a Markov Decision Process where fire is the agent at any cell in the landscape and the goal is to learn a policy for fire spreading into neighbouring cells. The set of suitable actions the fire can take from a location at any point of time includes spreading North, South, East, West or stay. Rewards are provided at the end of the epoch based on correctly classifying cells which are on fire or not. We apply two Reinforcement Learning algorithms to this problem: Value Iteration and Asynchronous Advantage Actor-Critic (A3C), which is a recent direct policy search approach that utilizes Deep Learning to perform simultaneous state-space approximation and policy representation.

The data for the start state and rewards come solely from satellite images of a region in northern Alberta, Canada, which is prone to large wildfires. Two events are used, the Fort McMurray fire of 2016 which led to the unprecedented evacuation of almost 90,000 people for several months and the Richardson fire of 2011 which was larger and more dramatic. Experiments are carried out training a wildfire spread policy for one region on multiple time frames as well as testing the ability to apply the learned policy to data from a second region. The results obtained indicate that it is useful to think of fire as a learning agent to understand its characteristics in a spatial environment.

Keywords: Value iteration; Asynchronous Advantage Actor-Critic; Satellite images; Forest Wildfires; Spatiotemporal Model; Spatial information; Deep Reinforcement Learning.

1 Introduction

The risk, costs and impacts of forest wildfires are a perennial and unavoidable concern in many parts of the world. A number of factors contribute to the increasing importance and difficulty of this domain in future years including climate change, growing urban sprawl into areas of high wildfire risk and past fire management practices which focused on immediate suppression at the cost of increased future fire risk [9].

There are a wide range of challenging decision and optimization problems in the area of Forest Fire Management [7], many of which would benefit from more responsive fire behaviour models which could be run quickly and updated easily based on new data. For example, the simplest decision problem could be whether to allow a fire to burn or not, which requires a great deal of expensive simulation to evaluate fully [5].

The field of forest and wildland fire behaviour modelling is a very active one involving trials in real forest conditions, lab experiments, physics-based fire modelling and more [3]. A challenge in the field is balancing the detail and computational cost of a physics-based model with the high level of accuracy needed for important decision making and planning problems. In this work we are contributing to that effort by investigating how Reinforcement Learning(RL) could be used to model wildfire spread as an agent on the landscape taking spatial actions in reaction to its environment.

Previous work on the application of reinforcement learning to spatial processes [4] has investigated modelling the state variable to represent the land cover and environmental characteristics and the action variable to represent interaction between characteristics of the neighbourhood geographic space. We apply these general principles in this work to wildfire prediction.

This work has similarities to the use of intelligent systems for predicting burned areas as suggested in [1]. However, their work focused on burned area alone, where as we look at the more specific problem of predicting the actual fire spread location over the short term. Their work also relies on high resolution field data which may not be possible to obtain for all fires. Our study uses easily accessible satellite data from large number of satellite data providers. A summary [6] of the merits of various software packages and tools (such as GeoMISC, GeoKNN, GeoMLP etc) in relation to Machine Learning on Geospatial Data was useful in developing this work.

2 Design and Experiments

The problem is formulated as a Markov Decision Process (MDP) $< S, A, P, R >$ where the state is each location on the landscape and the 'agent' taking actions is a fire spreading across the landscape. A state $s \in S$ corresponds to the state of a cell in the landscape (x, y, t, l, w, d, b) where x, y are the location of the cell, t is the temperature and l is the land cover value of the cell (derived from satellite images), and w and d are wind speed and direction. The state element b records whether the cell has been burned by fire ($b = 1$) or not ($b = 0$) or whether this is uncertain from the satellite images ($b = 2$). The initial state will come from satellite images that correspond to the beginning of a fire. Since we are focusing on fire growth rather than ignition detection we set certain cells to have just ignited fire and assign these to the initial state.

The reward function R is applied to a state and maps one of three values to each cell. A cell damaged by the wild fire is assigned a reward value of 1, cells clearly undamaged by the fire are assigned -1, while all the other cells are initialized with a reward value of 0. The action is then, at each instant, for the fire to 'choose' one of the North, South, East or West or stay actions to take from the current cell. The goal of the agent is to choose its actions so as to maximize the rewards gained within a time constraint. Wind Speeds and Wind Direction are taken into consideration before an action is taken. These values are assumed to be a constant for a small region of consideration at a fixed point of time. The dynamics for any particular cell $P(s'|s, a)$ is a function from one state s to the next s' given an action a. In this formulation the dynamics are actually quite simple as most aspects of the cell state do not change quickly, or do not change at all. So the action of spreading a fire into a cell directly alters the chance that neighbouring cells will move to a burn state. The fire is made capable of taking one of the possible actions from each cell. The fire is constrained to not cross the boundary of the domain of study. Figure 1 shows a representation of the model. The light/green cells are unburned and dark/red are those cells affected by fire. The circle represents an ignition point while at each action choice, the current cell decides to spread to a neighbouring cell or not.

2.1 Using Satellite Data

The study area under consideration is a northern part of the province Alberta, Canada (53.9333N, 116.5765 W). The satellite images are downloaded from the USGS Earth Explorer data portal (https://earthexplorer.usgs.gov/) for Alberta. Figure 2(b) shows an example where the smoke over burning areas can be seen. A series of images corresponding to the duration of occurrence of the Fort McMurray fire (approx. May 1 2016 to August 1 2016) and Richardson fire (approx. May 15, 2011 to September 2011) were collected for this region of Alberta. All the images were corrected for missing values and outliers. Additional pre-processing steps were carried out as outlined in [2]. For the region under

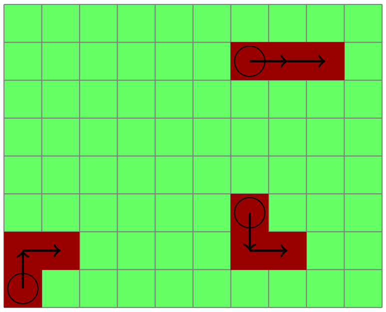
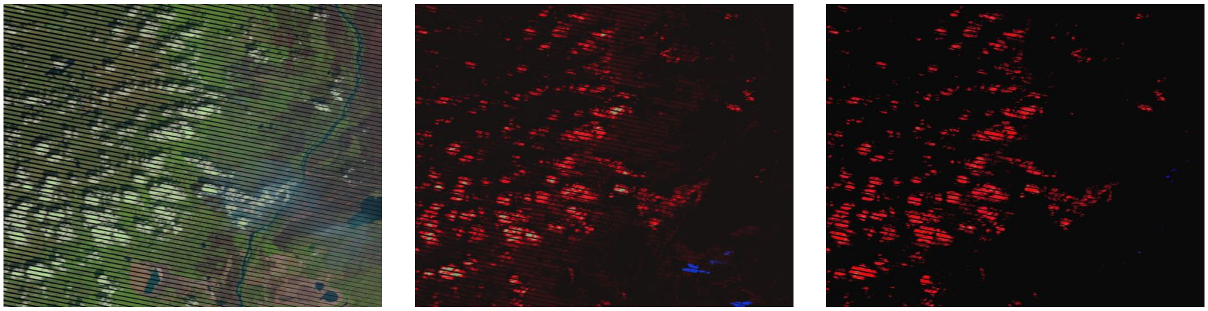


Figure 1: A schematic of the wildfire motion domain.



(a) The satellite image of june 8, 2011 in the region of Alberta

(b) Output obtained from value iteration

(c) Output obtained from deep learning

Figure 2: The wildfire spread domain as a grid model representing input satellite data and output wildfire prediction

consideration the original 7611×8401 pixel images are divided into distinct cells of 100×100 pixels which are used to represent a single cell state.

The land cover value is obtained by processing the satellite images in an open source geo-processing software (Multispec). The affected state of a cell is also obtained from the satellite image. Temperature is obtained from processing thermal images from the same data source. Wind speed and wind direction are processed from images of STAR data portal (https://manati.star.nesdis.noaa.gov/datasets/ASCATData.php) for the region of study.

2.2 Solution 1 : Value Iteration Method

For the first attempt at this problem we use asynchronous value iteration where the optimal value of the state $V^*(s)$ under the greedy fire spread policy is given by the Bellman Equation:

$$V^*(s) = R(s) + max_a \gamma \sum_{s'} P(s'|s,a) V^*(s') \tag{1}$$

where s' is the successor state and γ denotes the discount factor which we set to 0.99.

2.3 Solution 2: Deep Learning Method

The second algorithm we tried on this domain was the **Asynchronous Advantage Actor Critic (A3C)** algorithm [8]. In this algorithm a global network is defined in addition to multiple worker agents with individual sets of parameters. The input to this global network was formalized as a grid of 100 X 100 cells with each cell having state values which is an average of the state values of several pixels derived from satellite images. Here, once again, the reinforcement learning problem is formulated as the fire being the agent and deciding to spread from one cell to another. In $A3C$ we have the advantage of defining multiple worker agents. Each separate instance of a fire (unconnected to other fires) in a neighbourhood is given its environment as an input and the fire is defined as an individual worker. 15 instances of fire (thus, 15 worker agents) were considered for training and testing. Each worker (fire) would then update the global environment and we have plotted the result obtained. The deep network used is based on that given in [8] which uses an input layers of 100×100 pixel windows from the satellite image for the start date. Then there is an convolution layer of 16 88 filters with a stride of 4 follow by a rectifier nonlinear unit (ReLU) layer. The second hidden layer involves 32

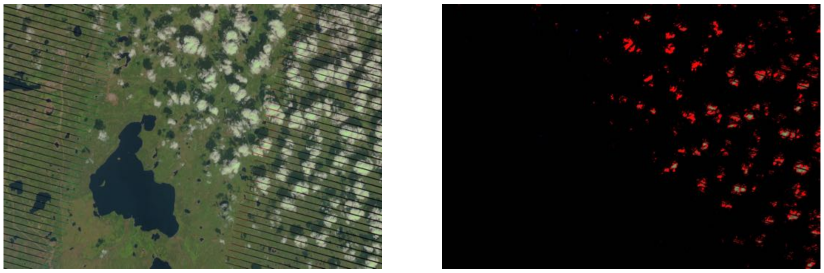

(a) The satellite image of june 14, 2016 in the region of Alberta
(b) Output obtained from value iteration

Figure 3: Comparing the output from value iteration to the actual scenario in case of Fort McMurray fire

4×4 filters using a stride of 2 also followed by a ReLU layer. The final fully-connected layer uses 256 ReLUs which is output to a layer of threshold functions for the output of each of the five possible actions.

2.4 Experimental Setup

In the first two experiments, (A) and (B), satellite images from May, June and July 2011 of the Richardson wildfire were chosen. The May images were taken right from before the ignition of the fire and used as the start state for RL. The satellite images in June (A) and July (B) were used to determine the target burned area and so inform the reward function. During training, the predicted final burned areas at the end of the period were used to compute the reward for the training epoch. The values obtained as a result were used to obtain a value function that predicts the growth and characteristics of fire after fixed time durations. For testing, the output from the value iteration was compared to an actual satellite image from halfway through the training period in the same manner.

In the third experiment (C), we applied the policy learned on the Richardson fire to satellite images of a different fire in Fort McMurray five years later. This is a similar region of Northern Alberta (but not exactly the same area) so the general properties of the policy should be applicable but these particular images were never part of the training set. The ignition location and initial spread are taken from the satellite images of the month of May. As before, the burned areas predicted by the learned policy a month after ignition were compared with the true satellite data. The three experiments were carried out using value iteration and the $A3C$ method. The convergence threshold was fixed to be 0.1 for all experiments.

3 Results

All experiments were run on an Intel core i7-6700 cpu with 32Gb ram. The value iteration approach took an average of 2.5 hours to converge while the A3C solution took around 4 hours. Figure 2(b) shows the results obtained from the experiment (A). The red in the output image corresponds to the pixels which were on fire and were classified correctly as fire by the algorithm (ie. true positives). The blue pixels represent false positives, those which were classified as burning but were not yet at that point. White pixels represent false negatives where the policy predicted no fire but fire was indeed present. Black pixels represents true negatives, the pixels which were not on fire and rightly classified. It can be seen visually that the value iteration algorithm was able to capture most of the large fire zones of the Richardson fire on June 8. Still some of the smaller ignitions were not captured by the value iteration.

Figure 3 shows the results from the third experiment (C). The day of comparison was on June 14, 2016. Again we can see that the value iteration algorithm did fairly well capturing the major fire spots.

Figure 2(c) shows the results obtained from the Deep Learning method. The deep learning method performs slightly better than a simple value iteration as some small fire instances are also being identified by $A3C$ in addition to the ones identified by value iteration technique. The comparison was done on the same day (June 8, 2011) for Richardson fire.

The ROC curve (plot between ratio of false positive and ratio of true positives) in Figure 4 shows the effect of varying the threshold used for classifying if a pixel is affected by fire or not. The threshold was chosen to be 0.83 for the value iteration method and 0.7 for the Deep Learning $A3C$ method. This is because these thresholds correspond to high degree of true positives with only a small number of false positives.

Table 1 presents the accuracy (as a percentage) obtained in both the methods across different experiments. Columns (A) and (B) test the policy's prediction on the same fire event, where the test point is 1 month or 2 months after ignition, where the reward comes from a point a month later. Column (C) is testing the policy's prediction on a completely different fire five years later, in the same region, predicting a fire one month after ignition.

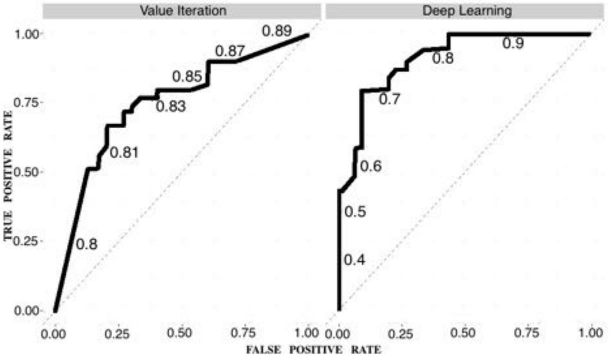


Figure 4: ROC curve for analysing sensitivity of policy to classification threshold parameter.

Method	(A) Richardson Fire(1st Month)	(B) Richardson Fire(2nd Month)	(C) Fort McMurray Fire
Value Iteration	79.6%	77.5%	70%
Deep Learning	87.3%	83.2%	81.8%

Table 1: Accuracy of each algorithm on the three different test scenarios.

4 Conclusion

In this work, we presented a novel approach for utilizing RL to augment physics-based forest wildfire simulations by learning the spread dynamics directly from readily available satellite image data. We demonstrated that a simple agent based model of fire can obtain reasonable fire spread predictions using value iteration and that this can be significantly improved using a Deep RL representation. In future work we plan to look at enriching the model by including more land characteristics such as moisture, slope and directional aspect as state variables in individual cells. We will also perform a wider comparison against different algorithms such as those in [1]. We plan to investigate improvements to the structure of the Deep Neural Network policy representation tailored to this kind of spatially spreading problem.

References

[1] M Castelli, L Vanneschi, and A Popovic. Predicting burned areas of for-est fires: an artificial intelligence approach. *Fire ecology*, 11(1):106–118, 2015.

[2] Matthew J Cracknell and Anya M Reading. Geological mapping using remote sensing data: A comparison of five machine learning algorithms, their response to variations in the spatial distribution of training data and the use of explicit spatial information. *Computers & Geosciences*, 63:22–33, 2014.

[3] Mark A. Finney, Jack D. Cohen, Sara S. McAllister, and W. Matt Jolly. On the need for a theory of wildland fire spread. *International Journal of Wildland Fire*, 22(1):25–36, 2013.

[4] N Forsell, F Garcia, and R Sabbadin. Reinforcement learning for spatial processes. In *18th World IMACS/MODSIM Congress*, pages 755–761, 2009.

[5] Rachel M. Houtman, Claire A. Montgomery, Aaron R. Gagnon, David E. Calkin, Thomas G. Dietterich, Sean McGregor, and Mark Crowley. Allowing a wildfire to burn: Estimating the effect on future fire suppression costs. *International Journal of Wildland Fire*, 22(7):871–882, 2013.

[6] M Kanevski, A Pozdnukhov, and V Timonin. Machine learning algorithms for geospatial data. applications and software tools. *Integrating Sciences and Information Technology for Environmental Assessment and Decision Making*, 1:320–327, 2008.

[7] David L Martell. A Review of Recent Forest and Wildland Fire Management Decision Support Systems Research. *Current Forestry Reports*, 1(2):128–137, 2015.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

[9] Claire Montgomery. Fire: An Agent and a Consequence of Land Use Change. In Joshua M. Duke and JunJie Wu, editors, *The Oxford Handbook of Land Economics*, chapter 13, pages 281–301. Oxford University Press, 2014.

[10] Andrea Woo and Carrie Tait. Up to 90,000 evacuated from Fort McMurray; some ordered to move again as fire spreads. *The Globe and Mail*, May 2016.

Goal-directed planning in a two-player game

Bas van Opheusden
Center for Neural Science
New York University
New York, NY 10003
basvanopheusden@nyu.edu

Gianni Galbiati
Center for Neural Science
New York University
New York, NY 10003
gianni.galbiati@nyu.edu

Zahy Bnaya
Center for Neural Science
New York University
New York, NY 10003
zahy.bnaya@nyu.edu

Yunqi Li
Center for Neural Science
New York University
New York, NY 10003
yunqi.li@nyu.edu

Wei Ji Ma
Center for Neural Science &
Department of Psychology
New York University
New York, NY 10003
weijima@nyu.edu

Abstract

Goal-directed decision-making algorithms commonly consist of two components: a rule to describe how people learn state values through experience, and a rule by which they choose actions based on learned values. Recent studies of human reinforcement learning have used increasingly sophisticated learning rules while often assuming simple decision rules. However, sequential decision-making tasks such as chess or go require model-based planning inside the decision rule. In this article, we propose a computational model for human decision-making in a challenging variant of tic-tac-toe. The model assigns state values based on simple positional features, and plans actions by searching a decision tree. We compare the model to various alternatives, and use it to study individual skill differences as well as the effects of time pressure and the nature of expertise. Our findings suggest that people perform less tree search under time pressure, and that players search more as they improve during learning.

Keywords: Sequential decision-making, Behavioral modeling, Expertise

Acknowledgements

This work was supported by grant IIS-1344256 from the National Science Foundation.

Introduction

Although the computations underlying human reinforcement learning are extensively studied, the process by which people explore decision trees is less understood. Most work focuses on the neural implementation of learning and decision-making in small decision trees [1, 2]. However, with more choices and more available options, the decision tree grows exponentially, and people need to prune the tree [3].

In this article, we investigate how people plan actions in a two-player board game. We develop a computational model that predicts people's choices on individual moves, and fit this model to data from individual participants. To validate the model, we test how its parameters change when we introduce time constraints on participant's moves. Finally, we use the model to investigate how people's decision process changes as they acquire expertise.

Experiments

Task. We collected data from people playing a variant of tic-tac-toe, in which players need to make 4-in-a-row on a 4-by-9 board (figure 1). Despite these simple rules, the game is surprisingly challenging and fun to play. Because the game is deterministic, contains no hidden information and the state space is reasonably small, it is solvable in theory and practice. With perfect play, the first player wins.

Procedure. First, we conducted a "human-vs-human" experiment with 40 participants, divided into 20 pairs. Participants in each pair played games against each other without time constraints for 50 minutes, switching colors every game. Next, we conducted an experiment with 30 participants, who played against computer opponents with a time limit of 5, 10 or 20 seconds per move. The time limit was constant within games and varied randomly across games. If participants exceeded the time limit, they lost the game. The computer opponents implemented an early version of our computational model for people's decision-making process, with parameters adapted from fits on human-vs-human games. We created 30 AI agents, grouped by playing strength into 6 groups of 5 agents each, and matched participants with AI opponents through a one-up, one-down staircase procedure. Finally, we conducted a third experiment (30 participants), who played against computer opponents without time limits for 5 sessions, spaced no more than 3 days apart.

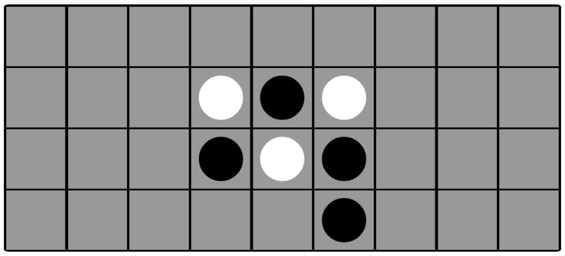


Figure 1: Task. Two players take turns placing black or white pieces on a 4-by-9 board, and the first to achieve 4-in-a-row (horizontal, diagonal or vertical), wins the game.

Model

Value function. The core component of our model is an evaluation function $V(s)$ which assigns values to board states s. We use a weighted linear sum of 5 features: center, connected 2-in-a-row, unconnected 2-in-a-row, 3-in-a-row and 4-in-a-row. The center feature assigns a value to each square, and sums up the values of all squares occupied by the player's pieces. This value of each square is inversely proportional to its Euclidean distance from the board center. The other features count how often particular patterns occur on the board (horizontally, vertically, or diagonally):

Connected 2-in-a-row: two adjacent pieces with enough empty squares around them to complete 4-in-a-row.

Unconnected 2-in-a-row: two non-adjacent pieces which lie on a line of four contiguous squares, with the remaining two squares empty.

3-in-a-row: three pieces which lie on a line of four contiguous squares, with the remaining square empty. This pattern represents an immediate winning threat.

4-in-a-row: four pieces in a row. This pattern appears only in board states where a player has already won the game.

We associate weights w_i to these features, and write

$$V(s) = c_{\text{self}} \sum_{i=0}^{4} w_i f_i(s, \text{self}) - c_{\text{opp}} \sum_{i=0}^{4} w_i f_i(s, \text{opponent})$$

where $c_{\text{self}} = C$ and $c_{\text{opp}} = 1$ whenever the player is to move in state s, and $c_{\text{self}} = 1$ and $c_{\text{opp}} = C$ when it is the opponent's move. The scaling constant C captures value differences between "active" and "passive" features. For example, a three-in-a-row feature signals an immediate win on the player's own move, but not the opponent's.

Tree search. The evaluation function guides the construction of a decision tree with an iterative best-first search algorithm. Each iteration, the algorithm chooses a board position to explore further, evaluates the positions resulting from each legal move, and prunes all moves with value below that of the best move minus a threshold. After each iteration, the algorithm stops with a probability γ, resulting in a geometric distribution over the total number of iterations.

Noise. To account for variability in people's choices, we add three sources of noise. Before constructing the decision tree, we randomly drop features (at specific locations and orientations), which are omitted during the calculation of $V(s)$ anywhere in the tree. During tree search, we add Gaussian noise to $V(s)$ in each node. Finally, we include a lapse rate λ.

Methods

Estimating task performance. To quantify task performance in human-vs-computer games, we use the Elo rating system [4], which estimates playing strength from game results, independent of the moves played. We append the results of games from all 4 experiments to a computer-vs-computer tournament, and estimate ratings jointly for all humans and computers with a Bayesian optimization algorithm [5].

Estimating model parameters. The model has 10 parameters: the 5 feature weights, the active-passive scaling constant C, the pruning threshold, stopping probability γ, feature drop rate δ and the lapse rate λ. We infer these parameters for individual participants and individual learning sessions or time limit conditions with maximum-likelihood estimation. We estimate the log probability of a participant's move in a given board position with inverse binomial sampling [6], and optimize the log-likelihood function with multilevel coordinate search [7]. To account for potential overfitting, we report 5-fold cross-validated log-likelihoods, with the same testing-training splits for all models.

Model comparison

To test how well our model predict participants' choices, we compare its log-likelihood on human-vs-human games to that of 26 alternative models (figure 2). We test four categories of alternative models: lesions, generated by removing model components; extensions, generated by adding new model components; modifications, generated by replacing a model component with a similar implementation; and controls, which are structurally different from the main model.

Lesions. We create lesion models by forcing either one of the feature weights to zero, or removing the feature dropping, pruning, value noise, active-passive scaling or the decision tree. The no-tree model evaluates the positions after each possible move, and chooses the one with maximum value. It contains feature dropping and value noise but no pruning.

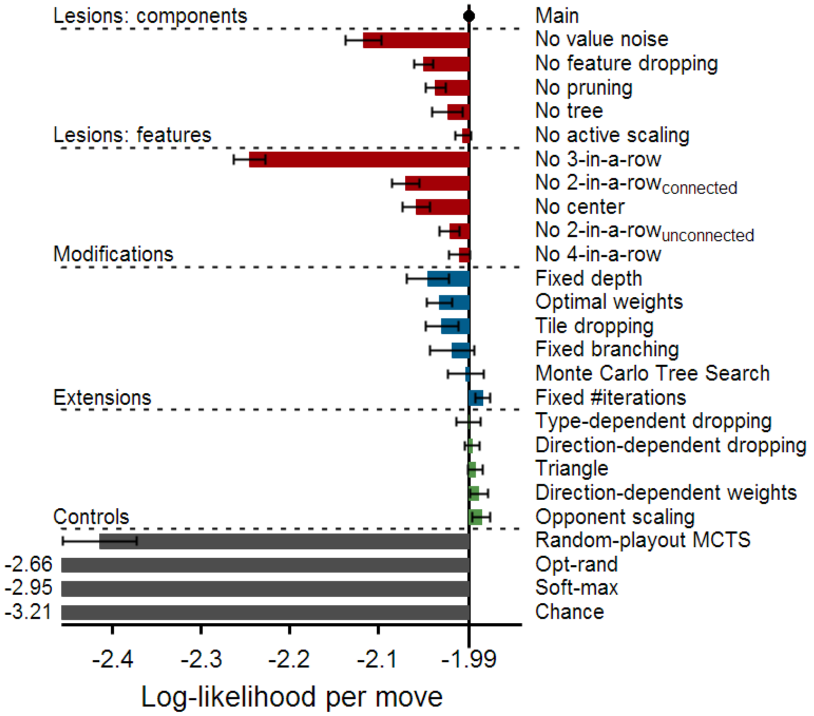


Figure 2: Cross-validated log-likelihood/move for our main model and 26 alternatives on the human-vs-human data. The bars show mean and s.e.m. across participants ($N = 40$). The main model fits better than lesions, most controls and some modifications, and approximately equally good as extensions or some other modifications.

Extensions. We consider extending the model with a feature that recognizes a three-piece pattern arranged in a triangle, or multiplying the weights for diagonally and vertically oriented features by scaling constants c_{diag} or c_{vert}, respectively. Alternatively, we extend the main model by allowing feature drop rates to differ between features of different types (2-in-a-row, 3-in-a-row, etc) or orientations. Finally, we test a model in which all weights for the opponent's features are scaled by a factor c_{opp}, controlling the balance between attack and defense.

Modifications. We modify the model by fixing the number of iterations of the search algorithm to a constant instead of the geometric distribution prescribed by the main model. Alternatively, we amend the search process to explore each branch of the tree up to fixed depth, or the pruning rule to keep only the K best moves (according to the evaluation function), where the branching factor K is again fixed. For a more drastic modification, Monte Carlo Tree Search (MCTS) estimates state values not by calling the evaluation function $V(s)$, but by aggregating outcomes of simulated games between no-tree agents. It also extends the best-first search algorithm by adding a term that favors exploration (investigating unexplored moves) over exploitation (further investigating already explored moves). We consider fixing the feature weights to the optimal solution, i.e. those weights that maximize the correlation between $V(s)$ and the game-theoretic value of the position s. Finally, we modify the attention mechanism from dropping random features from the evaluation function to dropping random branches from the decision tree.

Controls. We consider MCTS with completely random playouts, or a mixture model between optimal and random play. The optimal agent enumerates all candidate moves that preserve the game-theoretic value of the position, and chooses randomly between them. Another control model, labeled soft-max, assigns a value to each square on the board (enforced to obey reflection/rotation symmetry), and chooses a move with a softmax decision rule, constrained to unoccupied squares.

The full model fits better than all lesions and most modifications. The Monte Carlo Tree Search model is equally good and the "fixed iterations" model slightly outperforms it. All extensions similarly increase the model's log-likelihood a little bit, and all control models fit much worse than the main model. Unfortunately, the model comparison does not reveal a unique best-fitting model, meaning that we did not collect enough data to determine precise details of people's thought process. Specifically, we cannot distinguish between different tree search algorithms (best-first search or MCTS) or particulars of the best-first search algorithm (pruning and number of iterations). Alternatively, different participants may use different strategies. However, the model comparison does suggest that any model that can predict human choices with a limited number of free parameters needs to contain a feature-based evaluation function, and mechanisms for attentional oversights and tree search.

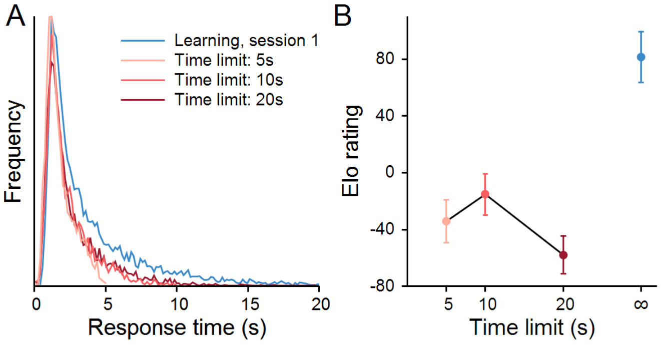


Figure 3: **A.** Histogram of response times in the three conditions of the time pressure experiment (red) and the first session of the learning experiment (blue). In the latter experiment, players could take arbitrary amounts of time, which we denote as an infinite time limit. **B.** Task performance, quantified by Elo rating, for the same conditions. Error bars indicate mean and s.e.m. across participants ($N = 30$). The effect of time limits on performance is unclear.

How experimental manipulations affect model parameters

To further support the model, we investigate whether its parameters respond in predictable ways to experimental manipulations. As our first manipulation, we introduce time constraints of 5, 10 or 20 seconds per move. Second, we conduct an experiment in which participants play the game for 5 sessions.

Given a set of parameters for an individual participant in a time limit condition or learning session, we simulate moves made by the model in a database of predetermined positions and measure 3 statistics of its process: the percentage of dropped features, the value quality (correlation between $V(s)$ and the game-theoretic value $V^*(s)$) and the mean tree size (number of nodes in its decision tree). Note that tree size incorporates both the width and depth of the decision tree.

Time pressure To test the effectiveness of time constraints to manipulate participants' behavior, we first plot the distribution of response times in the three conditions (figure 3A) as well as the first session of the learning experiment, where thinking time was unconstrained. Adding time pressure causes an overall shift in the response time distribution regardless of the time limit. Additionally, participants play faster with shorter time constraints. Surprisingly, there is no consistent effect of time constraints on participants' performance (figure 3B).

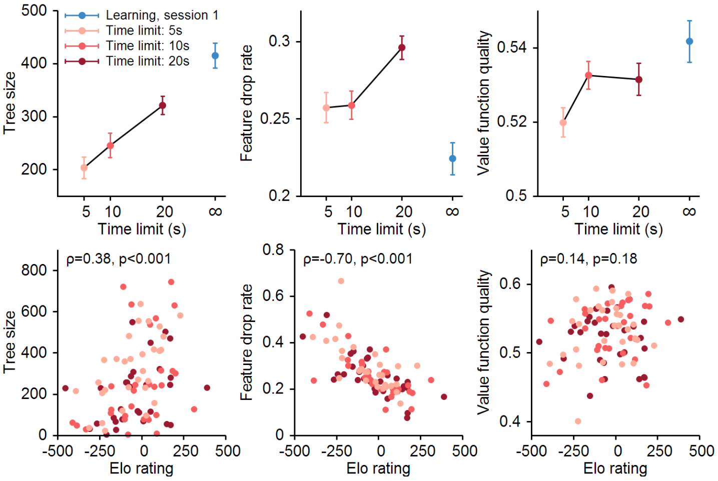


Figure 4: **Top row**. Estimated model parameters in the time pressure experiment. Error bars denote mean and s.e.m. across participants. The model infers a relation between time limit and tree size, but unclear effects on feature dropping and the value function quality. **Bottom row**. Model parameters and Elo rating for each participant in each time limit condition. The tree size and feature drop rate correlate with Elo rating, but value function quality does not.

In figure 4 (top), we show the feature drop rate, value function quality and tree size in different time limit conditions. Compared to the unconstrained experiment, participants build smaller trees and drop more features, while the value function quality is similar. The impact of the time constraint on tree size becomes larger with shorter time limits, but the feature drop rate shows the opposite trend and is at its highest in the 20-second condition. We speculate that the stress of potentially losing on time causes participants to pay more attention with shorter time limits, whereas with 20 seconds, they are more relaxed and make more attentional lapses.

To understand the surprising negative result of figure 3, we compute the correlation between Elo rating and parameter estimates across both individuals and time limit conditions (figure 4, bottom). Stronger players (in all time limit conditions) build larger decision trees and drop fewer features. Therefore, the increased tree size with longer time limit predicts a performance increase, but the increased feature drop rate predicts decreased performance. These opposite effects happen to be roughly equal, which explains why Elo rating and time limit don't correlate.

Learning We first validate that experience affects participants' behavior by plotting Elo rating as a function of session number (figure 5). Next, we show that across sessions, tree size increases, feature drop rate decreases and value function quality remains constant (figure 6, top). As in the time pressure experiment, tree size and feature drop rate correlate with Elo rating on an individual level (figure 6, bottom), and the change in parameter estimates across sessions explains changes in task performance. Experienced players build larger decision trees and drop fewer features, both of which lead to increased playing strength, which matches the data. Note that in the time pressure experiment, both tree size and feature dropping increase across conditions, whereas in the learning experiment, tree size increases while feature dropping decreases. Therefore, these behavioral characteristics are dissociable, and the previous results cannot be explained away by parameter trade-offs.

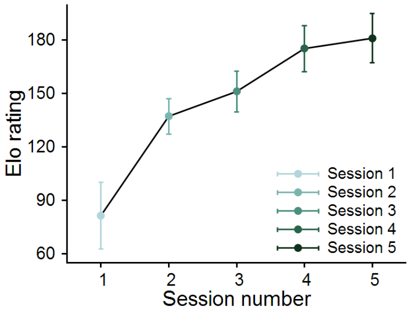


Figure 5: Elo rating of $N = 30$ participants in the learning experiment (mean and s.e.m. across participants). As participants gain expertise, they play stronger.

Conclusion

We built a computational model that predicts people's choices in a two-player game. The model posits three computational principles for sequential decision-making: a feature-based evaluation function, attentional oversights and tree search. All three components are necessary to explain participants' behavior, but the data does not constrain details of their implementation such as the order by which nodes are visited during search, or how long the search process continues before players finalize their decision.

The feature drop rate and tree size change in predictable ways when we expose participants to manipulations in time pressure and experience. These changes account for participants' task performance, suggesting that these specific parameters reflect some task-relevant characteristics of participants' cognitive process. In the future, we aim to further support our model as a description of the computational process underlying people's choices by using it to predict response times and eye movements.

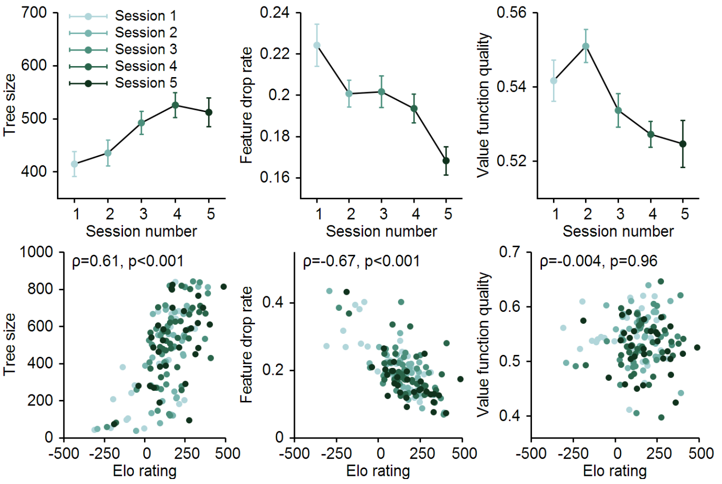


Figure 6: **Top**: Model parameters as a function of sessions completed in the learning experiment. Over the course of learning, tree size is estimated to increase while feature dropping decreases. The value function quality decreases, but only slightly. **Bottom**: Model parameters and Elo ratings for each participant in each session of the learning experiment. Both tree size and feature dropping correlate with Elo, but value function quality does not.

References

[1] Alec Solway and Matthew M Botvinick. Evidence integration in model-based tree search. *Proceedings of the National Academy of Sciences*, 112(37):11708–11713, 2015.

[2] Dylan Alexander Simon and Nathaniel D Daw. Neural correlates of forward planning in a spatial decision task in humans. *Journal of Neuroscience*, 31(14):5526–5539, 2011.

[3] Quentin JM Huys, Neir Eshel, Elizabeth O'Nions, Luke Sheridan, Peter Dayan, and Jonathan P Roiser. Bonsai trees in your head: how the pavlovian system sculpts goal-directed choices by pruning decision trees. *PLoS Comput Biol*, 8(3):e1002410, 2012.

[4] Arpad E Elo. *The rating of chessplayers, past and present.* Arco Pub., 1978.

[5] David R Hunter. Mm algorithms for generalized bradley-terry models. *Annals of Statistics*, pages 384–406, 2004.

[6] Morris H de Groot. Unbiased sequential estimation for binomial populations. *The Annals of Mathematical Statistics*, pages 80–101, 1959.

[7] Waltraud Huyer and Arnold Neumaier. Global optimization by multilevel coordinate search. *Journal of Global Optimization*, 14(4):331–355, 1999.

Assessing the Potential of Computational Modeling in Clinical Science

Peter F. Hitchcock
Department of Psychology
Drexel University
pfh26@drexel.edu

Angela Radulescu
Department of Psychology
Princeton University
angelar@princeton.edu

Yael Niv
Department of Psychology
Princeton University
yael@princeton.edu

Chris R. Sims
Department of Psychology
Drexel University
crs346@drexel.edu

Abstract

There has been much recent interest in using reinforcement learning (RL) model parameters as outcome measures in clinical science. A prerequisite to developing an outcome measure that might co-vary with a clinical variable of interest (such as an experimental manipulation, intervention, or diagnostic status) is first showing that the measure is stable within the same subject, absent any change in the clinical variable. Yet researchers often neglect to establish test-retest reliability. This is especially a problem with behavioral measures derived from laboratory tasks, as these often have abysmal test-retest reliability. Computational models of behavior may offer a solution. Specifically, model-based analyses should yield measures with lower measurement error than simple summaries of raw behavior. Hence model-based measures should have higher test-retest reliability than behavioral measures. Here, we show, in two datasets, that a pair of RL model parameters derived from modeling a trial-and-error learning task indeed show much higher test-retest reliability than a pair of raw behavioral summaries from the same task. We also find that the reliabilities of the model parameters tend to improve with time on task, suggesting that parameter estimation improves with time. Our results attest to the potential of computational modeling in clinical science.

Keywords: computational psychiatry, psychometrics, test-retest reliability, individual differences, trial-by-trial modeling

Introduction

There has been much recent interest in translating reinforcement learning (RL) tasks into assays for use in clinical science (Hitchcock, Radulescu, Niv, & Sims, 2017; Maia & Frank, 2011). Clinical science studies typically seek to test whether some parameter or set of parameters (such as computational modeling parameters, or statistical summaries of raw behavior) co-vary with an independent variable. The independent variable might be an experimental manipulation, an intervention, or a difference in diagnostic status. For example, a large analysis of an RL task found depression and anhedonia (independent variables) were both associated with lower values of a reward sensitivity parameter (an outcome measure) (Huys, Pizzagalli, Bogdan, & Dayan, 2013).

To show that a potential outcome measure co-varies in a meaningful way with an independent variable, it is first necessary to establish that the measure is stable within-subject in the absence of change in the independent variable. That is, the measure must show high test-retest reliability. If the test-retest reliability of a candidate outcome measure is low, it is unlikely that it cleanly samples some latent construct, such as a cognitive or learning process, that is stable within individuals. A low-reliability measure is unlikely to have much long-term utility in clinical science, as it is likely to be too noisy to meaningfully and reliably co-vary with independent variables of interest (Rodebaugh et al., 2016).

Despite the importance of establishing the test-retest reliability of potential outcome measures, researchers often neglect to do this. Neglect is especially common for behavioral measures in laboratory tasks, even though such measures often have abysmal test-retest reliability (Lilienfeld, 2014). An (in)famous example is the dot probe task (Hitchcock et al., 2017). The task was used for decades, in dozens of clinical studies, before it was shown to have close to 0 test-retest reliability (Price et al., 2015; Schmukle, 2005). The measure's lack of stability likely explains continued, widespread replication failures in studies employing it. More generally, the low reliability of measures from many laboratory paradigms poses a serious threat to progress in clinical science (see Rodebaugh et al., 2016 for discussion).

Traditionally, the measures derived from laboratory paradigms have been statistical summaries of raw behavior. For example, the outcome measure typically used in the dot probe task is average reaction time. Yet it has long been known that these measures tend to be highly variable and imprecise (Mischel, 1968). In addition to the fact that models can expose variables that are latent in behavior (e.g., learning rate), one source of excitement about applying RL modeling in clinical science comes from the idea that modeling behavior trial-by-trial will allow for the creation of outcome variables with lower measurement error than simple summaries of raw behavior (Hitchcock, 2017; Huys, Maia, & Frank, 2016).

One consequence of decreased measurement error should be more stability of measures at test and retest. However, no empirical study (of which we are aware) has compared head-to-head the test-retest reliability of RL model parameters with measures summarizing raw behavior. Thus, we compared the test-retest reliability of model parameters and behavioral summary measures derived from the Dimensions Task (Leong, Radulescu, et al., 2017; Niv et al., 2015; Radulescu et al., 2016), a multidimensional bandit with high selective attention demands. Radulescu et al., 2016 (study 2) found large differences between older adults and younger adults on two measures—one behavioral (accuracy; $g = .94$) and the other a computational modeling parameter (*decay*; older adults median = .52, younger adults median = .42)—suggesting the task has promise as a sensitive measure of individual differences.

Methods

We compared the test-retest reliability of two behavioral and model parameter measures derived from the Dimensions Task in two datasets. Datasets are from Niv et al. (2015; hereafter **D1**) and Radulescu et al. (2016, study 2; herafter **D2**).

Specifications. The datasets differed in total trials as well as the number of trials that comprised a game. (In each "game" one dimension of the bandits determined reinforcement probability; reinforcement contingencies reset and the participant had to learn contingencies anew in each game; see Niv et al., 2015.) In D1 ($N = 22$), subjects played 500 trials (number of trials per game was drawn from a Uniform(15,25) distribution, for a total of M=22.27, SD=1.45 games per subject). In D2 ($N = 54$), subjects played ~1400 trials (M=46.43, SD=5.41 games; subjects stopped playing after exactly 40 minutes; all games were 30 trials).

Measures. Following Radulescu et al. (2016), the behavioral measures were accuracy (trials with a correct response/total trials) and number of games learned (a game was defined as learned if the participant selected the most rewarding bandit on 6 consecutive trials). The model parameters came from an RL model with decay of weights of unchosen stimuli, and were d (decay rate) and $\beta*\eta$ (the product of inverse temperature and the learning rate). We used $\beta*\eta$ as a single parameter because we have previously found in these data that β and η are highly correlated (Hitchcock et al., 2017), consistent with known identifiability issues between inverse temperature and learning rate parameters in RL models (e.g., Schönberg et al., 2007). When we used $\beta*\eta$, the four measures were only modestly correlated in both datasets (Figure 1a). For more details on the Dimensions Task, including the computational model (known as *feature RL + decay*) and its free parameters, see Niv et al. (2015). See Hitchcock et al. (2017) and Radulescu et al. (2016) regarding the clinical potential of the task.

Test-retest reliability. Test-retest reliability of behavioral and computational modeling measures was assessed via 'A-1' intraclass (ICC) correlation coefficients (McGraw & Wong, 1996). ICCs were calculated on parameter fits and behavioral statistics by splitting the data into first (test) and second (retest) halves, and then calculating the ICC for each measure across these halves. Each half consisted of a set of games. Of note, because reward contingencies were reset in each game, each game (and hence also each half) was independent. Also of note, the "halving" was approximate because of the task's structure into games; specifically, the halfway split was made at the first game change after half the total trials had elapsed. A high ICC (that is, a higher correlation between first and second half scores for a given measure) reflects high within-subject stability in the measure.

Results and Discussion

In support of the premise that RL modeling can yield more precise, stable measures of individual differences than summaries of raw behavior (see also, Hitchcock, 2017), model parameters (d and $\beta^*\eta$) showed higher test-retest reliability than measures summarizing raw behavior (accuracy and number of games learned) in both datasets (Figure 1b). Specifically, whereas the ICCs of the behavioral measures in both datasets were nearly 0, consistent with ICCs of other laboratory behavioral measures such as measures from the dot-probe task (Price et al., 2015; Schmukle, 2005), the ICCs of the model parameters were much higher. Notably, in D2, the parameter with the highest test-retest reliability (d) outperformed the behavioral measure with the highest test-retest reliability (accuracy) by a factor of more than 4 (.68 versus .16).

Of note, the test-retest reliability of both model parameters was significantly higher in D2 than D1. Two differences in the task specifications could have led to this difference: (1) each game had more trials in D2 (30/game compared to an average of 20/game in D1); (2) many more games were played in D2 (~46/participant compared to ~22/participant in D1). Each of these changes could conceivably have enhanced reliability in D2. To better understand the difference responsible for the increase, we computed in D2 test-retest reliabilities for pairs of game subsets of varying length (range: 10-18), with the games in each subset drawn randomly and without replacement from the first and second halves of the task (i.e., 10-18 games per half from the first and second half). For example, the two vectors of game draws from the first and the second halves of the task for a given participant in a subset in which n games were drawn might look like: vector 1: {1, 2, 5, … , 21}; vector 2: {24, 25, 28, … , 47}. Since this procedure of randomly drawing a

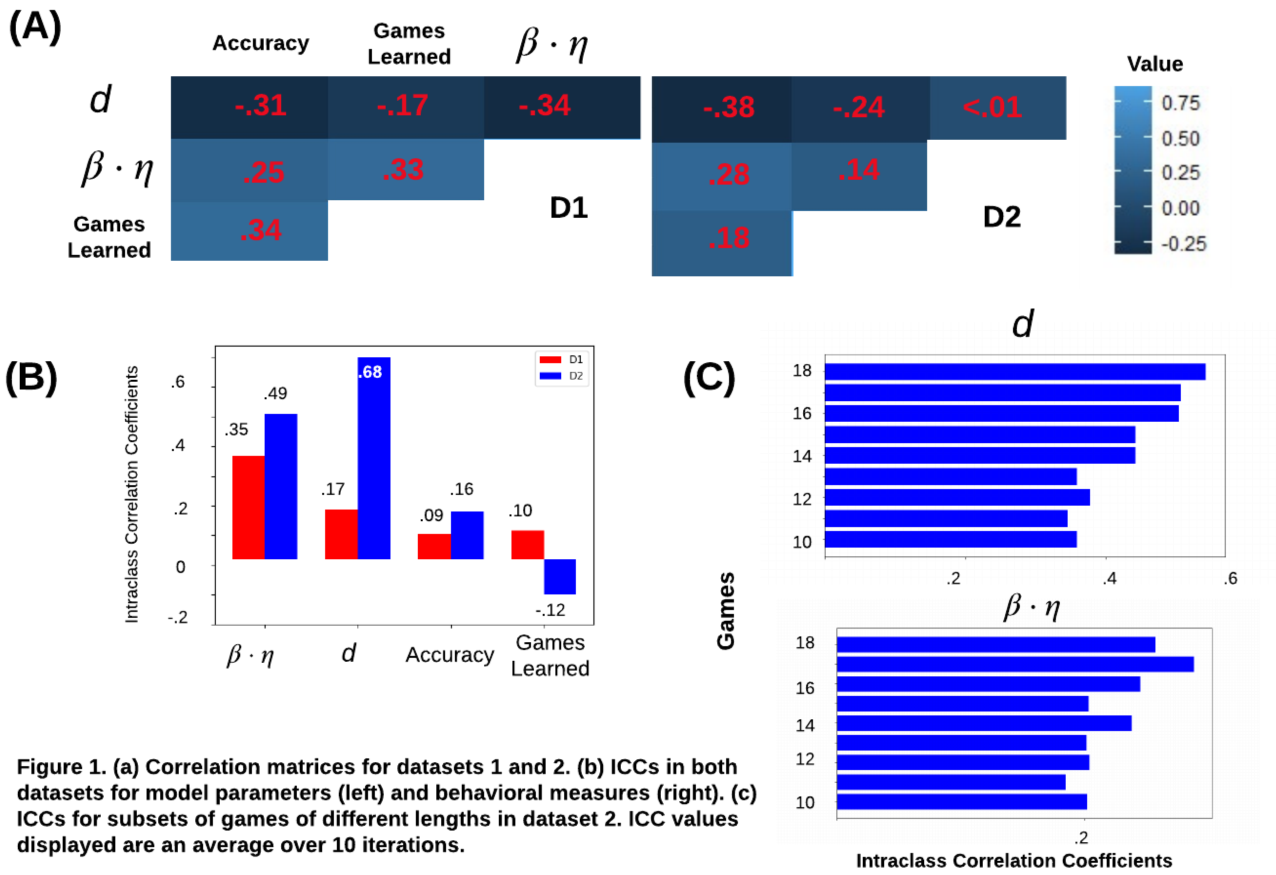


Figure 1. (a) Correlation matrices for datasets 1 and 2. (b) ICCs in both datasets for model parameters (left) and behavioral measures (right). (c) ICCs for subsets of games of different lengths in dataset 2. ICC values displayed are an average over 10 iterations.

subset of games entails drawing arbitrary games, we resampled games 10 times for each game length. On each iteration, the model was fit to the games in the subset, parameter estimates were derived, and an ICC score was calculated on these estimates. Figure 1c shows the average ICC at each game length.

The top graph of Figure 1c shows that the *d* parameter tended to improve with number of games played. Notably, when the number of games played in each subset was approximately equal to the number of games played in D1 (10-12 games per half), the ICCs range from only .35-.38. This is still higher than the ICC estimate for D1 (.17), in which a comparable number of games were played (~11 per half). But it is much lower than the ICC estimate for D2 when all games in the dataset were fit (.68; ~23 games per half). This suggests that the number of games played is a major factor driving the reliability of this parameter. For $\beta * \eta$ (bottom graph, Figure 1c), there also appears to be some improvement in reliability as the number of games played increases, but the improvement is much weaker than for the *d* parameter. In future work, we plan to explore further the factors that increase the reliability of $\beta * \eta$.

General Discussion

Overall, the higher reliabilities of the model parameters relative to behavioral measures support the promise of computational modeling for clinical science. As modeling improves, model parameters may ultimately exhibit test-retest reliability comparable to that of questionnaire measures. Questionnaire measures are the current standard-bearers in test-retest reliability (Mischel, 1968; Lilienfeld, 2014), but their *validity* is marred by response biases inherent to questionnaires (Hitchcock, 2017; Podsakoff et al., 2003). Thus, it is noteworthy that, at least for one parameter under one set of task specifications (*decay* in D2), we found that test-retest reliability (.68) approached a level considered adequate for questionnaires. This level of reliability is almost unheard of for a measure derived from behavior, particularly when the measure is only collected at a single visit (Epstein, 1979; Lilienfeld, 2014). These early results suggest the possibility that, as computational psychiatry progresses and modeling approaches are refined, model parameters may eventually achieve the holy grail of psychometrics: high reliability *and* high validity.

Acknowledgements

This work was supported by ARO grant W911NF-14-1-0101 (YN), NIMH grant R01MH098861 (YN & AR), and NSF research grant DRL-1560829 (CRS).

References

Epstein, S. (1979). The stability of behavior: I. On predicting most of the people much of the time. *Journal of Personality and Social Psychology*, *37*(7), 1097-1126.

Hitchcock, P.F. (2017). Computational Modeling and Reform in Clinical Science. [preprint; osf.io/mvxkf] *Open Science Framework*.

Hitchcock, P.F., Radulescu, A., Niv, Y., Sims, C.R. (2017). Translating a Reinforcement Learning Task into a Computational Psychiatry Assay: Challenges and Strategies. In *Proceedings of the 39th Annual Meeting of the Cognitive Science Society*.

Huys, Q. J., Pizzagalli, D. A., Bogdan, R., & Dayan, P. (2013). Mapping anhedonia onto reinforcement learning: a behavioural meta-analysis. *Biology of mood & anxiety disorders*, *3*(12), 1-16.

Huys, Q. J., Maia, T. V., & Frank, M. J. (2016). Computational psychiatry as a bridge from neuroscience to clinical applications. *Nat. Neuro.*, *19*(3), 404-413.

Leong, Y. C., Radulescu, A., Daniel, R., DeWoskin, V., & Niv, Y. (2017). Dynamic Interaction between Reinforcement Learning and Attention in Multidimensional Environments. *Neuron*, *93*(2), 451-463.

Lilienfeld, S. O. (2014). The Research Domain Criteria (RDoC): an analysis of methodological and conceptual challenges (pp. 13-14). *Beh. Res. and Ther.*, *62*, 129-139.

Maia, T. V., & Frank, M. J. (2011). From reinforcement learning models to psychiatric and neurological disorders. *Nature Neuroscience*, *14*(2), 154-162.

McGraw, K. O., & Wong, S. P. (1996). Forming inferences about some intraclass correlation coefficients. *Psychological methods*, *1*(1), 30-46

Mischel, W. (1968). *Personality and Assessment*. New York: Wiley.

Niv, Y., Daniel, R., Geana, A., Gershman, S. J., Leong, Y. C., Radulescu, A., & Wilson, R. C. (2015). Reinforcement learning in multidimensional environments relies on attention mechanisms. *J. of Neurosci.*, *35*(21), 8145-8157.

Podsakoff, P. M., MacKenzie, S. B., Lee, J. Y., & Podsakoff, N. P. (2003). Common method biases in behavioral research: a critical review of the literature and recommended remedies. *Journal of applied psychology*, *88*(5), 879-903.

Price, R. B., Kuckertz, J. M., Siegle, G. J., Ladouceur, C. D.,... & Amir, N. (2015). Empirical recommendations for improving the stability of the dot-probe task in clinical research. *Psych. Assessment*, *27*(2), 365-376.

Radulescu, A., Daniel, R., & Niv, Y. (2016). The effects of aging on the interaction between reinforcement learning and attention. *Psychology and Aging*, *31*(7), 747-757.

Rodebaugh, T. L., Scullin, R. B., Langer, J. K., Dixon, D. J., Huppert, J. D., Bernstein, A., Zvielli, A., & Lenze, E. J. (2016). Unreliability as a threat to understanding psychopathology: The cautionary tale of attentional bias. *Journal of Abnormal Psychology*, *125*(6), 840-851.

Schmukle, S. C. (2005). Unreliability of the dot probe task. *European Journal of Personality*, *19*(7), 595-605.

Schönberg, T., Daw, N. D., Joel, D., & O'Doherty, J. P. (2007). Reinforcement learning signals in the human striatum distinguish learners from nonlearners during reward-based decision making. *Journal of Neuroscience*, *27*(47), 12860-12867.

Scalability of Deep Reinforcement Learning for Cooperative Control

Jayesh K. Gupta
Department of Computer Science
Stanford University
`jkg@cs.stanford.edu`

Maxim Egorov
Department of Aeronautics and Astronautics
Stanford University
`megorov@stanford.edu`

Mykel J. Kochenderfer
Department of Aeronautics and Astronautics
Stanford University
`mykel@stanford.edu`

Abstract

This work considers the problem of learning cooperative policies in complex, partially observable domains without explicit communication. We extend three classes of single-agent deep reinforcement learning algorithms based on policy gradient, temporal-difference error, and actor-critic methods to cooperative multi-agent systems. To effectively scale these algorithms past a trivial number of agents, we combine them with a multi-agent variant of curriculum learning. The algorithms are benchmarked on a suite of cooperative control tasks, including tasks with discrete and continuous actions, as well as tasks with dozens of cooperating agents. We report the performance of the algorithms using different neural architectures, training procedures, and reward structures. We show that policy gradient methods tend to outperform both temporal-difference and actor-critic methods and that curriculum learning is vital to scaling reinforcement learning algorithms in complex multi-agent domains.

Keywords: multi-agent, reinforcement learning, deep learning

1 Introduction

Learning to cooperate between several interacting agents has been well studied. While the problem of cooperation can be formulated as a decentralized partially observable Markov decision process (Dec-POMDP), exact solutions are intractable. A number of approximation methods for solving Dec-POMDPs have been developed recently that adapt techniques ranging from reinforcement learning to stochastic search. However, applying these methods to real-world problems is challenging because they are typically limited to discrete actions and require carefully designed features.

On the other hand, recent work in single agent reinforcement learning has enabled learning in domains that were previously thought to be too challenging due to their large and complex observation spaces. This line of work combines ideas from deep learning with earlier work on function approximation, giving rise to the field of deep reinforcement learning. Deep reinforcement learning has been successfully applied to complex real-world tasks that range from playing Atari games to robotic locomotion. The recent success of the field leads to a natural question — how well can ideas from deep reinforcement learning be applied to cooperative multi-agent systems?

In this work, we focus on problems that can be modeled as Dec-POMDPs. We extend three classes of deep reinforcement learning algorithms: temporal-difference learning using Deep Q Networks (DQN), policy gradient using Trust Region Policy Optimization (TRPO), and actor-critic using Deep Deterministic Policy Gradients (DDPG) and Asynchronous Advantage Actor-Critc (A3C). We consider three training schemes for multi-agent systems based on centralized training and execution, concurrent training with decentralized execution, and parameter sharing during training with decentralized execution. We incorporate curriculum learning into cooperative domains by first learning policies that require a small number of cooperating agents and then gradually increasing the number of agents that need to cooperate. The algorithms and training schemes are benchmarked on four multi-agent tasks requiring cooperative behavior. The benchmark tasks were chosen to represent a diverse variety of complex environments with discrete and continuous actions and observations.

Our empirical evaluations show that multi-agent policies trained with parameter sharing and an appropriate choice of reward function exhibit cooperative behavior without explicit communication between agents. We show that the multi-agent extension of TRPO outperforms all other algorithms on benchmark problems with continuous action spaces, while A3C has the best performance on the discrete action space benchmark. By combing curriculum learning and TRPO, we demonstrate scalability of deep reinforcement learning in large, continuous action domains with dozens of cooperating agents and hundreds of agents present in the environment. To our knowledge, this work presents the first cooperative reinforcement learning algorithm that can successfully scale in large continuous action spaces.

2 Cooperative Reinforcement Learning

2.1 Centralized

A centralized policy maps the joint observation of all agents to a joint action, and is equivalent to a MPOMDP policy. A major drawback of this approach is that it is centralized in both training and execution, and leads to an exponential growth in the observation and actions spaces with the number of agents. We address this intractability in part by factoring the action space of centralized multi-agent systems.

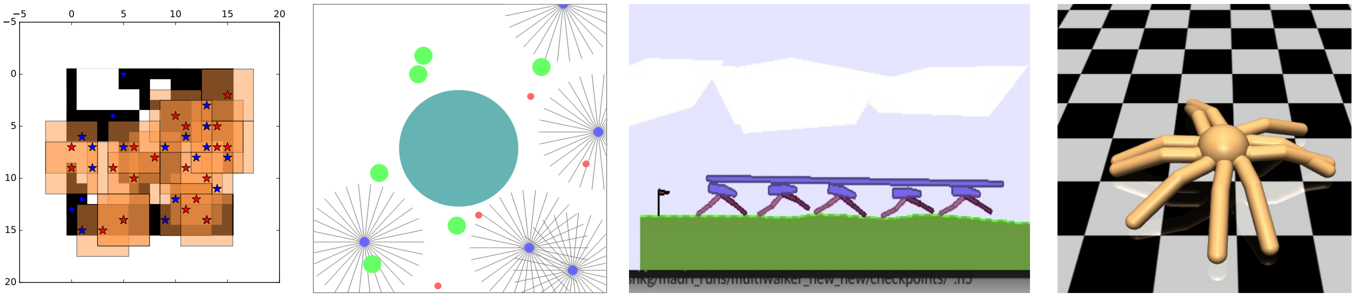
2.2 Concurrent

In concurrent learning, each agent learns its own individual policy. Concurrent policies map an agent's private observation to an action for that agent. Each agent's policy is independent.

The major drawback of concurrent training is that it does not scale well to large numbers of agents. Because the agents do not share experience with one another, this approach adds additional sample complexity to the reinforcement learning task. Another drawback of the approach is that the agents are learning and adjusting their policies individually making the environment dynamics non-stationary. This often leads to instability.

2.3 Parameter Sharing

The policies of homogeneous agents may be trained more efficiently using parameter sharing. This approach allows the policy to be trained with the experiences of all agents simultaneously. However, it still allows different behavior between agents because each agent receives unique observations, which includes their respective index. In parameter sharing, the control is decentralized but the learning is not. In the remainder of the paper, all training schemes are assumed to be parameter sharing unless stated otherwise. So long as the agents can execute these decentralized policies with shared parameters, single agent algorithms like DDPG, DQN, TRPO, and A3C can be extended to multi-agent systems.



(a) Pursuit (b) Waterworld (c) Multi-Walker (d) Multi-Ant

Figure 1: Examples of the four cooperative domains.

3 Tasks

Pursuit: Pursuit is a standard task for benchmarking multi-agent algorithms. The pursuit-evasion domain consists of two sets of agents: evaders and pursuers. The evaders are trying to avoid pursuers, while the pursuers are trying to catch the evaders. The action and observation spaces in this problem are discrete. Each pursuer receives a range-limited observation of its surroundings, and must choose between five actions Stay, Go East, Go West, Go South, Go North.

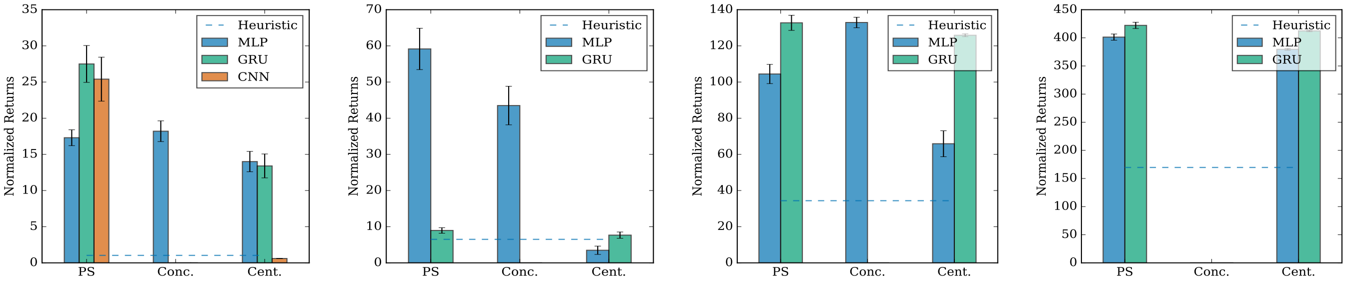
Waterworld: Waterworld can be seen as an extension of the above mentioned pursuit problem to a continuous domain. In this task, agents need to cooperate to capture moving food targets while avoiding poison targets. Both the observation and action spaces are continuous, and the agents move around by applying a two-dimensional force.

Multi-Walker: Multi-Walker is a more difficult continuous control locomotion task based on the BipedalWalker environment from OpenAI gym. The domain consists of multiple bipedal walkers that can actuate the joints in each of their legs. At the start of each simulation, a large package that stretches across all walkers is placed on top of the walkers. The walkers must learn how to move forward and to coordinate with other agents in order to keep the package balanced while navigating a complex terrain.

Multi-Ant: The multi-ant domain is a 3D locomotion task based on the quadrupedal robot used in. The goal of the robot is to move forward as quickly as possible. In this domain, each leg of the ant is treated as a separate agent that is able to sense its own position and velocity as well as those of its two neighbors. Each leg is controlled by applying torque to its two joints.

4 Experiments

4.1 Training Scheme



(a) Pursuit (b) Waterworld (c) Multi-Walker (d) Multi-Ant

Figure 2: Normalized average returns for multi-agent policies trained using TRPO. Missing entries indicate the training was unsuccessful. A random policy has zero normalized average return. Error bars represent standard error. The Wilcoxon test suggests the differences are significant ($p < 0.05$) except for the difference between centralized GRU and shared parameter GRU for the waterworld domain.

We first compared the performance of the three training schemes on the pursuit problem using TRPO. The results are summarized in Fig. 2. The figure shows that parameter sharing tends to outperform both the concurrent and centralized training schemes, especially when the number of agents in the environment increases.

4.2 RL Algorithm

We next compared the performance of different single agent deep reinforcement learning algorithms based parameter sharing approaches. As can be seen from Table 1, PS-A3C outperforms both PT-TRPO and PS-DQN, with PS-DQN having the worst performance for the discrete task. In the continuous domain tasks, we found that PS-TRPO outperforms both PS-A3C and PS-DDPG in the waterworld and multi-walker domains. We believe that with more hyperparameter tuning, PS-A3C results can be comparable as in the multi-ant environment.

We hypothesize that PS-DQN and PS-DDPG are unable to learn a good controller due to the changing policies of other agents in the environment. This makes the dynamics of the problem non-stationary which causes experience replay to inaccurately describe the current state of the environment.

Table 1: Average returns (over 50 runs) for policies trained with parameter sharing. DQN for discrete environment, DDPG for continuous

Task	PS-DQN/DDPG	PS-A3C	PS-TRPO
Pursuit	10.1 ± 6.3	25.5 ± 5.4	17.4 ± 4.9
Waterworld	NA	10.1 ± 5.7	49.1 ± 5.7
Multiwalker	-8.3 ± 3.2	12.4 ± 6.1	58.0 ± 4.2
Multi-ant	307.2 ± 13.8	483.4 ± 3.4	488.1 ± 1.3

4.3 Reward Structure

The concept of reward shaping involves modifying rewards to accelerate learning without changing the optimal policy. When modeling a multi-agent system as a Dec-POMDP, rewards are shared jointly by all agents. Decentralized representations allow us an alternative local reward representation. Local rewards can restrict the reward signal to only those agents that are involved in the success or failure at a task. An example would be to add a penalty restricted to the agent falling down in the Multi-Walker problem rather than distributing this penalty to all the agents because the team fails to carry the box forward. It has been shown that such local information can help reduce the number of samples required for learning. Table 2 confirms that local reward shaping leads to better performance, and is critical to learning intelligent behavior in multi-agent domains.

Table 2: Average returns for parameter sharing multi-agent policies with global and local rewards

Task	Global	Local
Pursuit	8.1	12.1
Waterworld	−1.4	14.3
Multi-Walker	−23.3	29.9
Multi-Ant	475.2	488.1

4.4 Scaling

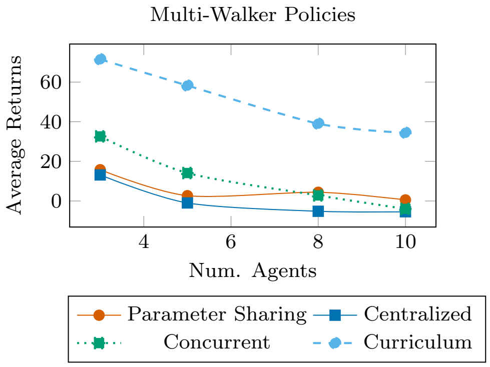


Figure 3: Performance of multi-walker policies as a function of the number of agents during training. Each data point in the shared parameters, centralized, and concurrent curves was generated by training and evaluating a policy with a fixed number of agents. The curriculum curve was generated by evaluating a single policy with varying numbers of agents.

Figure 3 shows the degrading performance of all policies with increasing number of agents in the multi-walker domain, and the performance improvements when curriculum learning is used. The policies were all trained with TRPO. The decrease in performance is in part due to the increasing difficulty of the reinforcement learning task as the number of cooperating agents grows. As the number of agents required to complete a task increases, it becomes more difficult to reach the parts of the state space with positive rewards using naive exploration policies.

We investigated how a curriculum learning scheme can help scale the multi-walker problem in the number of agents. An intuitive curriculum for this problem is over the number of agents, and so we define a curriculum with the number of agents in the environment ranging from 2 to 10. Because the policies are decentralized even though the parameters are shared, they can be evaluated on tasks with any number of cooperating agents regardless of the number of cooperating agents present during training. Unfortunately, we found that these decentralized shared parameter policies trained on a few agents often fail to generalize to larger numbers of agents.

To mitigate this, we define a Dirichlet distribution for this range of tasks with higher probability assigned to the simplest task (with two agents for the Multi-Walker domain). We then sample an environment from this distribution over the tasks in the curriculum and optimize the policy with PS-TRPO for a few iterations. Once the expected reward for the most likely environment reaches a threshold, we change the distribution such that the next environment is most likely. We continue this curriculum until the expected reward in all environments reaches the defined threshold. We believe this performance improvement is due to two reasons: a) The distribution over environments provides a regularization effect, helping avoid local minima during policy optimization, and b) A curriculum partially addresses the exploration problem by smoothly increasing the difficulty of the policy to be learned.

5 Conclusion

Despite the advances in decentralized control and reinforcement learning over recent years, learning cooperative policies in multi-agent systems remains a challenge. The difficulties lie in scalability to high-dimensional observation spaces and to large numbers of agents, accommodating partial observability, and handling continuous action spaces. In this work, we extended three deep reinforcement learning algorithms to the cooperative multi-agent context, and applied them to four high-dimensional, partially observable domains with many agents.

Our empirical evaluations show that PS-TRPO policies have substantially better performance than PS-DDPG and PS-A3C in continuous action collaborative multi-agent domains while PS-A3C is able to outperform PS-TRPO in the discrete domain. We suspect that DQN and DDPG perform poorly in systems with multiple learners due to the non-stationarity of the system dynamics caused by the changing policies of the agents. The non-stationary nature of the system makes experience replay samples obsolete and negatively impacts training. As evidence, we found that by disabling experience replay and instead relying on asynchronous training we were able to improve on the performance of DQN and DDPG. However, we believe more hyperparameter tuning might be required to reduce the gap in overall performance in continuous domains with respect to TRPO. Finally, we presented how cooperative domains can form a natural curriculum over the number of agents required to collaborate on a task and discovered how this not only allows us to scale PS-TRPO to environments with large number of cooperating agents, but owing to the regularization effect offered, allows us to reach better local optima in general.

There are several areas for future work. To improve scalability of the proposed approach for larger numbers of cooperating agents further future work is needed. Two major challenges in multi-agent systems are accommodating reward sparsity through intelligent domain exploration and incorporating high-level task abstractions and hierarchy. These are acute forms of similar challenges in the single agent learning. Recently, curiosity based information gain maximizing exploration strategy was explored by. Similar ideas could be adapted to maximize information gain not only about the environment's dynamics, but the dynamics of an agent's behavior as well. Correspondingly, hierarchical value functions were integrated with deep reinforcement learning. Incorporating task hierarchies in a multi-agent system would allow us to tackle learning specialization and heterogeneous behavior.

Prediction Regions and Tolerance Regions for Multi-Objective Markov Decision Processes

Maria Jahja
Department of Statistics
North Carolina State University
Raleigh, NC 27695
`mjahja@ncsu.edu`

Daniel J. Lizotte
Department of Computer Science
Department of Epidemiology & Biostatistics
University of Western Ontario
London, ON N6A 3K7
`dlizotte@uwo.ca`

Abstract

We present a framework for computing and presenting prediction regions and tolerance regions for the returns of a policy operating within a multi-objective Markov decision process (MOMDP). Our framework draws on two bodies of existing work, one in computer science for learning in MOMDPs, and one in statistics for uncertainty quantification. We review the relevant methods from each body of work, give our framework, and illustrate its use with an empirical example. Finally, we discuss potential future directions of this work for supporting sequential decision-making.

Keywords: decision support, multi-objective, prediction region

Acknowledgements

We are deeply indebted to XXX for their generous support of RLDM2017.

1 Introduction

Markov Decision Processes (MDP) [1] are useful conceptual tools for reasoning about sequential decision-making under uncertainty. Much of the computer science research on planning and learning in MDPs has focused on constructing an autonomous agent that acts in a given environment over an extended period of time, choosing actions according to a particular policy in order to achieve a high expected sum of rewards [12]. Other research, particularly in statistics, uses MDPs to frame the development of evidence-based decision support for sequential decision-making problems [2]. In the field of statistics, policies are often called *Dynamic Treatment Regimes* (DTRs), and there is a substantial literature studying their development and application in the field of health care decision-making [8, 11]. As in computer science, much of the literature is devoted to the estimation of regimes that optimize the expected sum of rewards, as well as to uncertainty quantification for the parameters and the performance of such regimes.

Most DTR literature focuses on the use of batch data to understand how treatments can be selected with the goal of achieving long-term success for a population of patients. Thus, the deployment of DTRs in statistics was always assumed to be "multi-agent" in the sense that the DTR would be used to aid the decisions of *many different patient-clinician pairs* where each patient would experience only one "episode" of the regime. Thus, there is a fundamental disconnect between the *average* performance of a regime ("How will a population respond?") and the *individual* performance of a regime ("How will an individual respond?"). In a computer science framework with a single agent over a long horizon, this difference matters much less; in decision support, however, recognizing the variability in performance achieved over individual episodes and communicating that variability to a human decision-maker is crucial. The difference can be viewed as one of statistical versus practical significance – with enough data, we may become extremely confident that one action affords higher *expected* return than another in a given state. However, if the variance of the returns is large, the question of which action will perform better in a particular episode may nearly come down to a fair coin flip.

In this work, our goal is to capture and convey information about the *distribution* of returns, rather than only the mean return, of a learned policy. We also wish to accommodate multiple reward signals; rather than formulate an MDP using a single, one-size-fits-all reward, we use a *Multi-Objective Markov Decision Process* (MOMDP) to enable the end user to consider several rewards that may be important (e.g. symptom relief, side effects, cost) before making their next decision.

To accomplish this goal, we extend and combine ideas from recent work on uncertainty quantification in MDPs and on Q-learning for MOMDPs. Lizotte and Tahmasebi [6] present methods for constructing *prediction intervals* and *tolerance intervals* for the returns of a policy. To a decision-maker their method conveys, given a state and action, what returns are likely to be observed, but is restricted to the single-reward setting. Lizotte and Laber [5] describe a methodological and computational framework for computing optimal policies of MOMDPs under different solution concepts using linear function approximation over a finite time horizon. Their approach, an extension of Q-learning, provides point estimates for the mean vector-valued returns achievable from a given state, but does not give information about the distribution of vector-valued returns.

Our main contribution is a framework for computing tolerance regions in the multiple-reward setting by augmenting both the tolerance interval algorithm from Lizotte and Tahmasebi [6] and the policy learning algorithm from Lizotte and Laber [5]. The output of our algorithm is the region of the space of returns that is most likely to contain a return achieved by following a non-dominated policy. We present a framework rather than a particular algorithm, because different applications are likely to warrant different components for Q-function approximation, solution concept, and tolerance region construction. We give a specific example to demonstrate how the framework functions but we also identify where components could be interchanged to suit a broad variety of tasks.

2 Background

In the following, we review background for Multi-Objective Markov Decision Processes and Q-learning in that context. We then review prediction regions and tolerance regions.

2.1 Multi-Objective Markov Decision Processes and Q-learning

This section follows the development in [5]. Human decision-making is often driven by multiple competing objectives; for example, a medical decision will be based not only on the effectiveness of a treatment, but also on its potential side-effects, cost, and other considerations. Because the relative importance of these objectives varies from user to user, the quality of a policy is not well captured by a universal single scalar "reward" or "value." Multi-Objective Markov Decision Processes (MOMDPs) accommodate this by allowing vector-valued rewards [5, 9] and using an application-dependent *solution concept* to define the performance of a policy. A solution concept is essentially a partial order on policies; the set of policies that are maximal according to the partial order are considered "optimal" and are indistinguishable under that solution concept – for example, using Pareto optimality as the solution concept leads to an equivalence class of all policies that lead to a value on the Pareto frontier. A collection of policies may be represented by a *non-deterministic* *policy* (NDP) [7]. Given an MDP with *state space* $\mathcal{S}$ and an *action set* $\mathcal{A}$, an NDP Π is a map from the state space to the set $2^{\mathcal{A}} \setminus \{\emptyset\}$. Like previous work [5, 9], we focus on the setting where the definition of an MDP is augmented by assuming a D-dimensional reward vector $\mathbf{R}(s_t, a_t)$ is observed at each time step. We define a MOMDP with finite time horizon T as a tuple of state spaces $\mathcal{S}_t$, action spaces $\mathcal{A}_t$, state transition functions $P_t : \mathcal{S}_t \times \mathcal{A}_t \to \mathbb{P}(\mathcal{S}_{t+1})$ where $\mathbb{P}(\mathcal{S}_{t+1})$ is the space of probability measures on $\mathcal{S}_{t+1}$, and reward functions $\mathbf{R}_t : \mathcal{S}_t \times A_t \to \mathbb{R}^D$ for $t \in \{1, ..., T\}$. In keeping with the Markov assumption, both $\mathbf{R}_t$ and P_t depend only on the current state and action. We assume finite action sets, but we do *not* assume that state spaces are finite. The *value* of a policy π is given by $\mathbf{V}^\pi(s) = \mathbb{E}^\pi[\sum_{t=1}^T \mathbf{R}^t(s_t, a_t) | s_1 = s]$, the expected sum of (vector-valued) rewards we achieve by following π.

Consider a batch of n trajectories $s_1^i, a_1^i, r_{1[1]}^i, ..., r_{1[D]}^i, s_2^i, a_2^i, r_{2[1]}^i, ..., r_{2[D]}^i, ..., s_T^i, a_T^i, r_{T[1]}^i, ..., r_{T[D]}^i$ for $i = 1, ..., n$. At time T, (the final time point) we define the approximate Q-function for reward dimension d as the least squares fit

$$\hat{Q}_{T[d]}(s_T, a_T) = \phi_T(s_T, a_T)^\intercal \hat{\mathbf{w}}_{T[d]}, \;\; \hat{\mathbf{w}}_{T[d]} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_i \left(\phi_T(s_T^i, a_T^i)^\intercal \mathbf{w} - r_{T[d]}^i \right)^2 \tag{1}$$

giving the estimated vector-valued expected reward function $\hat{\mathbf{Q}}_T(s_T, a_T) = (\hat{Q}_{T[1]}(s_T, a_T), ..., \hat{Q}_{T[D]}(s_T, a_T))^\intercal$. Here, $\phi_T(s_T, a_T)$ is a feature vector of state and action. Having obtained the $\hat{\mathbf{Q}}_T$ from (1), we construct an NDP Π_T that gives, for each state, the actions one might take at the last time point. For each state s_T at the last time point, each action a_T is associated with a *unique* vector-valued estimated expected reward given by $\hat{\mathbf{Q}}_T(s_T, a_T)$. Thus, we decide which among these vectors is a desirable outcome using our solution concept, and include their associated actions in $\Pi_T(s_T)$.

For $t < T$, it is only possible to define the expected return of taking an action in a given state by also deciding which particular policy will be followed to choose future actions. In standard fitted-Q, for example, one assumes that the future policy is given by $\pi_j(s) = \arg\max_a \hat{Q}_j(s, a)$ for all $j > t$. In the non-deterministic setting, we may know that the future policy belongs to some set of possible policies derived from Π_j for $j > t$, but in general we do not know which among that set will be chosen; therefore, we explicitly include the dependence of $\hat{\mathbf{Q}}_t$ on the choice of future policies $\pi_j, t < j \leq T$ by setting $\hat{\mathbf{Q}}_t(s_t, a_t; \pi_{t+1}, ..., \pi_T) = [\hat{Q}_{t[1]}(s_t, a_t; \pi_{t+1}, ..., \pi_T), ..., \hat{Q}_{t[D]}(s_t, a_t; \pi_{t+1}, ..., \pi_T)]^\intercal$ where for $d = 1, \ldots, D$, $\hat{Q}_{t[d]}(s_t, a_t; \pi_{t+1}, ..., \pi_T) = \phi_t(s_t, a_t)^\intercal \hat{\mathbf{w}}_{t[d]\pi_{t+1},...,\pi_T}$, and $\hat{\mathbf{w}}_{t[d]\pi_{t+1},...,\pi_T} = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^n [\phi_t(s_t^i, a_t^i)^\intercal \mathbf{w} - \{r_{t[d]}^i + \hat{Q}_{t+1[d]}(s_{t+1}^i, \pi_{t+1}(s_{t+1}^i); \pi_{t+2}, ..., \pi_T)\}]^2$.

We use $\mathcal{Q}_t$ to denote the *Q-pool*, a set of partially-evaluated Q-functions; each member of $\mathcal{Q}_t$ is a function of s_t and a_t only and assumes a particular fixed sequence $\pi_{t+1}, ..., \pi_T$ of future policies. Each one gives the estimated *expected* return for the given state-action pair and future sequence of policies. Our goal in this work is to augment each one with information about the distribution of those returns.

2.2 Prediction Regions and Tolerance Regions

A *prediction region* $\mathcal{R}^\alpha(Y_1, \ldots, Y_n)$ for a data generating process traps the next observation Y_{n+1} with probability $1 - \alpha$:

$$\Pr(Y_{n+1} \in \mathcal{R}^\alpha(Y_1, \ldots, Y_n)) = 1 - \alpha. \tag{2}$$

Conformal prediction (CP) is a methodology for constructing nonparametric prediction regions under mild assumptions [10]. CP methods produce valid prediction regions under any distribution using a given *nonconformity measure*. A nonconformity measure takes a data sample and produces *nonconformity scores* σ_i that measure the "novelty" of each observation with respect to the whole set. For example, we may define nonconformity in terms of regression predictions as $\sigma_i = |y_i - \widehat{y}_i|$ where $\widehat{y}_i = X\widehat{\beta}$ is the ith fitted value. Let F be the data generating distribution for $Y_1, \ldots, Y_n$. Once the nonconformity scores are obtained for all $i = 1 \ldots n$, the nonconformity σ_{n+1} of a hypothetical additional observation Y_{n+1} can be compared to the observed data to obtain a p-value for the null hypothesis $H_0 : Y_{n+1} \sim F$ by $p(Y_{n+1}; Y_1, \ldots, Y_n) = (n+1)^{-1} \sum_{i=1}^{n+1} \mathbf{1}\{\sigma_i \geq \sigma_{n+1}\}$. By definition, if $Y_{n+1} \sim F$, $\Pr(p(Y_{n+1}; Y_1, \ldots, Y_n) \leq \alpha) = \alpha$. Therefore, the region described by $\mathcal{R}^\alpha_{\text{CP}} = \{Y : p(Y; Y_1, \ldots, Y_n) > \alpha\}$ traps Y_{n+1} with probability $1 - \alpha$, and is a $1 - \alpha$ prediction region. Conformal prediction guarantees the produced prediction is valid under any given nonconformity measure–parametric or nonparametric– even in finite samples. However, the nonconformity measure does influence the size of the region and therefore its usefulness in practice. Lei, Robins, and Wasserman [3] propose the use of a kernel density estimate to produce the nonconformity score. We use their approach in our empirical example below.

A $(1 - \alpha)$, γ-content *tolerance region* $\mathcal{R}^{\alpha,\gamma}(Y_1, \ldots, Y_n)$ has the property

$$\Pr(P_F[\mathcal{R}^{\alpha,\gamma}(Y_1, \ldots, Y_n)] \geq \gamma) = 1 - \alpha. \tag{3}$$

Note that this is a much stronger probability statement than (2): it says that each interval we construct, with high probability, captures at least γ of the probability mass of the data generating distribution, F. Several parametric and non-parametric methods are known for one-dimensional tolerance intervals, and Lizotte and Tahmasebi demonstrate their use for constructing one-dimensional tolerance for the returns of estimated DTRs [6]. Li and Liu [4] propose a nonparametric method based on order statistics derived from *data depth*, which is a quantity very similar in spirit to a nonconformity measure. Tolerance intervals are then computed by applying one-dimensional techniques to data depth. Although this method produces tolerance regions, which make a stronger probability statement than prediction regions, (3) is shown to hold only asymptotically. Nonetheless, Li and Liu demonstrate that they can achieve good performance in some finite sample settings (i.e. with hundreds of data points and two dimensions).

3 A Framework for Prediction and Tolerance Regions for Multiple Objectives

We present a modular framework for constructing tolerance regions for multiple objectives. The set of optimal policies are found using methods from Lizotte and Laber [5], and the regions are constructed using methods from Lizotte and Tahmasebi [6]. Algorithm 1 lays out the major steps for constructing regions from policies learned by these methods.

The methods in [5] produce a *pool* of Q-functions at each timestep, as described above. For a given (s_t, a_t), each Q-function in the time t pool produces a different expected return, and its associated future policies produce a different distribution over returns. To construct a region (prediction or tolerance) for a particular Q-function from the pool, we identify the trajectories that start[1] with s_t, a_t and whose actions are consistent with that Q-function's assumed future policies. We then use the empirical distribution of their returns to construct whichever type of region is desired. Since in general we do not know which future policy will be followed, we propose to construct regions for all Q functions in the pool and then take their union.

Algorithm 1 Regions for MOMDPs

```
Input: Set of $n$ trajectories; region function $\mathcal{R}$; initial state action pair $(s_t, a_t)$
  Compute $Q$-pool for timestep $t$ [5]
  for each function in the pool do
    Collect all trajectories beginning with $(s_t, a_t)$ and consistent with the current $Q$-function
    If necessary, re-weight trajectories and/or add trajectories using residual borrowing [6]
    Construct a region by applying $\mathcal{R}$ to the returns of the resulting trajectories
  end for
  return The union of the regions
```

Lizotte and Tahmasebi note that if the exploration policy is allowed to depend on state, then the distribution of matched trajectories will not in general match the distribution of returns we would obtain by following the future policies under consideration [6]. Hence, constructing regions naïvely using the matched trajectories will yield incorrect results. They characterize this dependence and present an inverse probability weighting procedure for correcting the distribution before constructing regions. We propose to use the same strategy in the MOMDP setting when necessary. They also propose *residual borrowing*, which uses the residuals between the estimated Q values and the returns among the matched trajectories to infer what the distribution of returns would have been among the unmatched trajectories by combining them with the estimated Q-values. (For further detail please see reference [6].) This methodology increases the amount of trajectories we can use, and could also be used within our framework.

Our framework can accommodate any multivariate region method (prediction or tolerance) and any reweighting procedure (based on generalized additive models, density ratio estimation, etc.). In our opinion the most promising methods are the Lei-Robins-Wasserman conformal prediction method [3] and the Li-Liu tolerance region method [4].

4 Illustrative Example

To illustrate our methods, we use a generative model with 2 decision points, and consider a 2-dimensional reward vector which allows us to visualize the results. Our state-action pairs of interest are $(s_1, a_1) = (0, a_1)$, where A_1 is binary, S_2 is continuous, A_2 is binary, and Y is continuous. Drawing $n = 200$ trajectories, we split A_1 evenly between the actions 0 and 1. The generative model is defined as $S_2|S_1 = 0, A_1 = a_1 \sim \text{Normal}(\delta_{10} + \delta_{11}s_1 + \delta_{12}a_1 + \delta_{13}s_1a_1, 2)$. In this case $\delta_0 = (\delta_{10}, \delta_{11}, \delta_{12}, \delta_{13}) = (0, 0.5, -0.75, 0.25)$. $A_2 \sim \text{Bernoulli}(0.5)$. We generate the rewards by $Y_i|S_1 = 0, S_2 = s_2, A_1 = a_1, A_2 = a_2 \sim \text{Normal}(\mu_Y(s_1, s_2, a_1, a_2), 10)$, where $\mu_Y = \beta_{20} + \beta_{21}s_1 + \beta_{22}a_1 + \beta_{23}s_1a_1 + \beta_{24}s_2 + \beta_{25}s_2 + a_2(\psi_{20} + \psi_{21}a_1 + \psi_{22}s_2)$. Our model used $\beta_2 = (\beta_{20}, \beta_{21}, \beta_{22}, \beta_{23}, \beta_{24}, \beta_{25}) = (3, 0, 0.1, -0.5, -0.5, 0)$. For Y_1, $\psi_2 = (\psi_{20}, \psi_{21}, \psi_{22}) = (10, 0.4, 0.5)$, for Y_2, $\psi_2 = (1, 0.2, -1)$.

From these generated trajectories, we form prediction regions by following Algorithm 1 with a confidence level $1 - \alpha = 0.95$. Using the methods of Lizotte and Laber [5], we find 72 members for the Q-pool. For each member, we build a region by running conformal prediction with density estimation for the nonconformity score. In this example, we use a Gaussian kernel with bandwidth parameter selected using Scott's rule.

To visualize the shape of the boundary, we calculate the nonconformity scores for a 120×120 grid of sample points with respect to the trajectories where a_2 matches that of the policy. We calculate the test statistic for the conformal prediction hypothesis, and include sample points with scores that meet the level α. Figure 1 shows the results. In all panels, the data

---

[1] If S_t is continuous, we may need to select trajectories that start "near" s_t. The best way to do this is an open question.

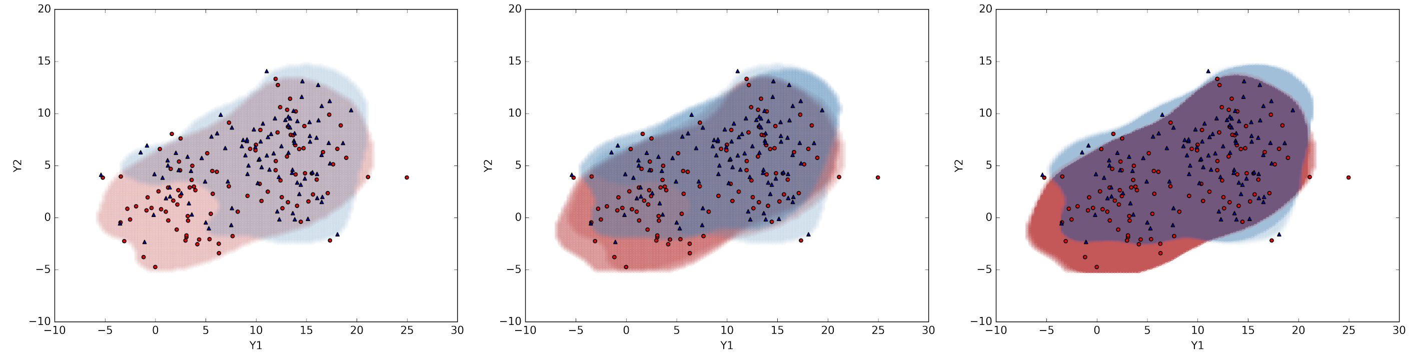


Figure 1: Data points are the observed returns, red dots for trajectories with $a_1 = 0$, blue triangles for those with $a_1 = 1$. *Left*: Prediction regions for one feasible policy. *Center*: Overlay of three prediction regions. *Right*: Prediction regions for all 72 feasible policies.

points are the 200 observed returns, red dots for $a_1 = 0$, blue triangles for $a_1 = 1$. In the leftmost panel, the prediction regions shown corresponds to the first feasible policy. Similarly to the point markers, the red-shaded region encloses the returns for trajectories with $a_1 = 0$, while the blue-shaded region corresponds to trajectories where $a_1 = 1$. The center panel overlays the prediction regions for the first three feasible policies for each action. On the right panel are the prediction regions corresponding to all 72 feasible policies.

5 Discussion and Conclusion

Our example shows how one might present and use the information from the regions: by overlaying the regions corresponding to different actions, we can compare their likely outcomes to aid decision-making. In forthcoming work, we will explore applications to medical treatment decision-making. In general, we expect that the best choices for the details of this framework (e.g. approximation method, region type, presentation style) will differ from task to task, and it is therefore our hope that this work provides a foundation for both novel research and useful application.

References

[1] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II*. Athena Scientific, 3rd edition, 2007.

[2] E. B. Laber, D. J. Lizotte, M. Qian, W. E. Pelham, and S. A. Murphy. Dynamic treatment regimes: technical challenges and applications. *Electronic Journal of Statistics*, 8(1):1225–1272, 2014.

[3] J. Lei, J. Robins, and L. Wasserman. Distribution-free prediction sets. *Journal of the American Statistical Association*, 108(501):278–287, 2013.

[4] J. Li and R. Y. Liu. Multivariate spacings based on data depth: I. construction of nonparametric multivariate tolerance regions. *The Annals of Statistics*, 36(3):1299–1323, 06 2008.

[5] D. J. Lizotte and E. B. Laber. Multi-objective markov decision processes for data-driven decision support. *Journal of Machine Learning Research*, 17(211):1–28, 2016.

[6] D. J. Lizotte and A. Tahmasebi. On prediction and tolerance intervals for Dynamic Treatment Regimes. *Statistical Methods for Medical Research*, 2017. 15 pages. Accepted.

[7] M. Milani Fard and J. Pineau. Non-deterministic policies in Markovian decision processes. *Journal of Artificial Intelligence Research*, 40:1–24, 2011.

[8] L. Orellana, A. Rotnitzky, and J. Robins. Dynamic regime marginal structural mean models for estimation of optimal dynamic treatment regimes, part i: Main content. *Int. Jrn. of Biostatistics*, 6(2), 2010.

[9] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[10] G. Shafer and V. Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9:371–421, March 2008.

[11] S. Shortreed, E. B. Laber, D. J. Lizotte, T. S. Stroup, J. Pineau, and S. A. Murphy. Informing sequential clinical decision-making through reinforcement learning: an empirical study. *Machine Learning*, 84(1–2):109–136, 2011.

[12] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Engagement matters: pupil and mental effort mediate depletion effect on subsequent physical tasks

Bryan Kromenacker, B.S.
Department of Psychology and
Cognitive Science Program
University of Arizona
bkromena@email.arizona.edu

Robert C. Wilson, Ph.D.
Department of Psychology and
Cognitive Science Program
University of Arizona
bob@email.arizona.edu

Abstract

Self-control depletion theory claims to account for between-task performance changes in terms of the consumption of a limited cognitive resource. Dual-task designs have been used to demonstrate that increased self-control on an initial effortful task predicted a decreased use of self-control on a later categorically distinct effortful task, suggesting a limited resource model. These accounts struggle to identify specific mechanisms linking them to rational theories of effort, and the reported effect size has recently come into question. Subject engagement during the depleting task is often assumed, but systematic disengagement may account for inconsistencies in the observed effect. We recreated a common dual-task depletion paradigm using a computer-automated design allowing for measurement of individual task performance as well as pupil size. We found evidence that task engagement measures do indeed account for some individual variation in the depletion effect, offering a possible explanation for inconsistent group-level effects.

Keywords: effort, decision-making, control, depletion, pupil

Acknowledgements

We thank Maxwell Alberhasky, Chrysta Andrade, Daniel Carrera, Kathryn Chung, Michael de Leon, Emily Giron, Brittany Gonzalez, Leah Hall, Maura Higgs, Min-Hee Kang, Kathryn Kellohen, Hannah Kyllo, Alex Lawwill, Stephanie Low, Colin Lynch, Genevieve Patterson, Filipa Santos, Shlishaa Savita, Catie Sikora, Vera Thornton, Guillermo Vargas, Christina West for help running the experiments.

1 Background

Why do we sometimes feel physically drained after a long day at the office? Going to the gym, cooking dinner, or even deciding what to watch on Netflix may feel more effortful than it does on the weekend. It seems intuitive that performance should decrease over time for physically strenuous tasks to prevent injury, but why should this be the case for mental performance? The physiological constraints of cognition are unknown, but tasks that engage executive functions are associated with a sense of mental effort. The perception of effort is now thought to be centrally generated (not an afferent feedback signal from metabolically strained tissue) [1]. Resource-depletion models capture a nonrational (i.e., not based upon explicit value calculation) aspect of performance deterioration on effortful tasks.

Resource-depletion theories explain performance reductions in terms of the "depletion" of some limited resource in proportion to the duration and magnitude of expended cognitive control [2,3]. This effect is tested using sequential task paradigms where performance reduction on task 2 is measured after differences in difficulty on task 1 [6,7]. Mechanistic accounts of limited metabolic resources [4,5] are challenged by observations that increasing the future reward for continuing a demanding task seems to mitigate the immediate performance drop. These findings support models that frame self-control as a type of limited *cognitive* resource. Performance reductions predicted by self-control depletion theory have now been supported by hundreds of studies, incorporating the use of conscious inhibition on effortful tasks demanding physical, mental, emotional, and appetitive control [8]. However, the theory has not provided clear neurophysiological or computational targets. In addition, a recent pre-registered, multisite trial of a classic dual-task pairing produced a null finding, reinforcing questions about effect size inflation from publication bias [9,10].

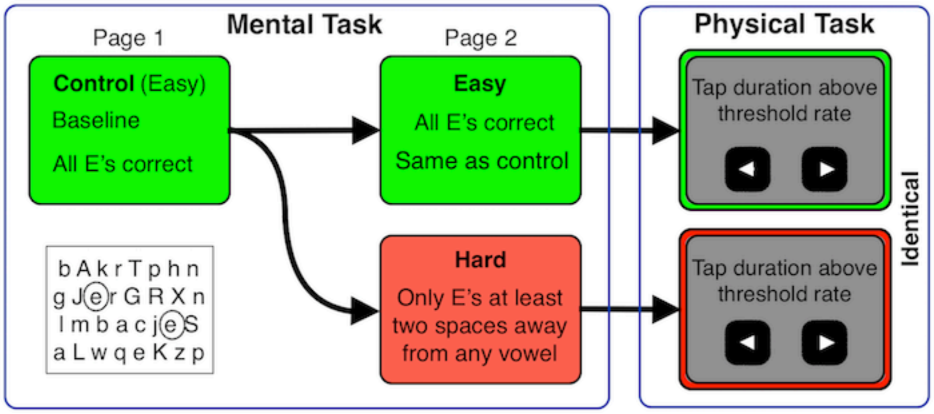
Due to inconsistent reports from depletion studies, we sought to recreate a classic self-control depletion paradigm using an automated format capturing individual changes in performance and engagement. Can we find a depletion effect in an automated paradigm? Accounting for individual differences in task engagement may reveal a reason for the inconsistent reports of the depletion effect. Is performance on the depleting task and overall engagement necessary to produce a depletion effect?

2 Methods

This study was designed to recreate the classic self-control depletion findings reported in the literature [8]. The depleting task, "crossing out E's", was chosen for the reported large effect size. All trials of the task were computer-automated using Matlab. Pupillometry data was recorded at 30 Hz using an Eye Tribe eye tracker. The dependent physical effort task was created to match the effort in squeeze-bar tasks, where subject must maintain a certain grip over time. Instead of a grip bar, this study required subjects to tap the left and right arrow keys on a keyboard as quickly as possible. This alternating, "running" motion of the fingers on one hand was maintained above an individually set threshold rate (70% of maximum individual tap rate) for as long as possible.

Depleting Task, Mental Effort. In the first crossing out "E's" task subjects were instructed to click on as many "E's" as possible on a page of random letters in paragraph format over 4 minutes (forming a habit of clicking all "E's"). This was followed by a second page where participants were randomized to either a control group (where the instructions were the same as the first page) or an experimental depletion group (where they were instructed to cross out "E" as long as it was not directly next to or one space away from any vowel—requiring inhibition of previously formed habit and holding rules in working memory). A circle appears around an "E"only after a correct click.

Dependent Measure, Physical Effort. The above task was followed by the dependent measure where participants were instructed to press the left and right arrow keys in an alternating fashion using only one hand. Participants' time above their individualized threshold rate was measured. Times above threshold were normalized by the threshold value. A difference in the total time that subjects performed above threshold was the main expected effect.

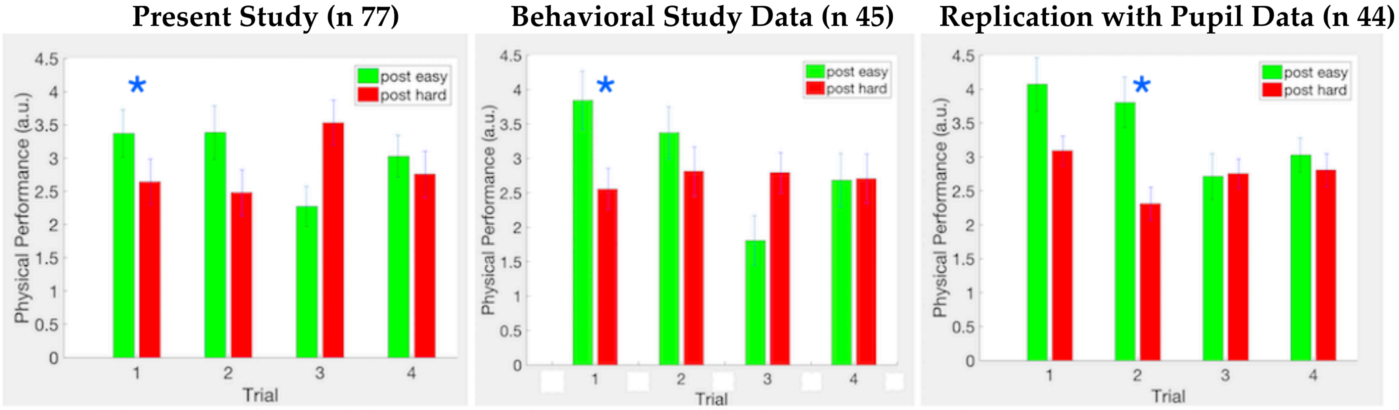


Subjects completed a repeated-measures crossover version of the dual-task paradigm. Condition grouping was randomly assigned to complete two trials of easy (control) tasks followed by two trials of hard (depleting) tasks or vice versa for a total of four trials for a 50-minute total time. A total of 77 subjects completed the study (38/39 per order condition).

3 Results and Discussion

3.1 Physical performance declines after difficult mental task, but only before crossover

The main predicted effect of decreased physical performance following difficult mental tasks was not present across all blocks, but was present between groups on the first trial. A trend in the predicted direction (greater performance after easy) was observed before the difficulty crossover, but not on trials after. This modest, but significant effect resembles the same trend seen in an earlier behavioral study (no pupillometry) and a replication study — no other null-finding studies or data have been excluded. Non-parametric one-way ANOVA was used to detect group difference.

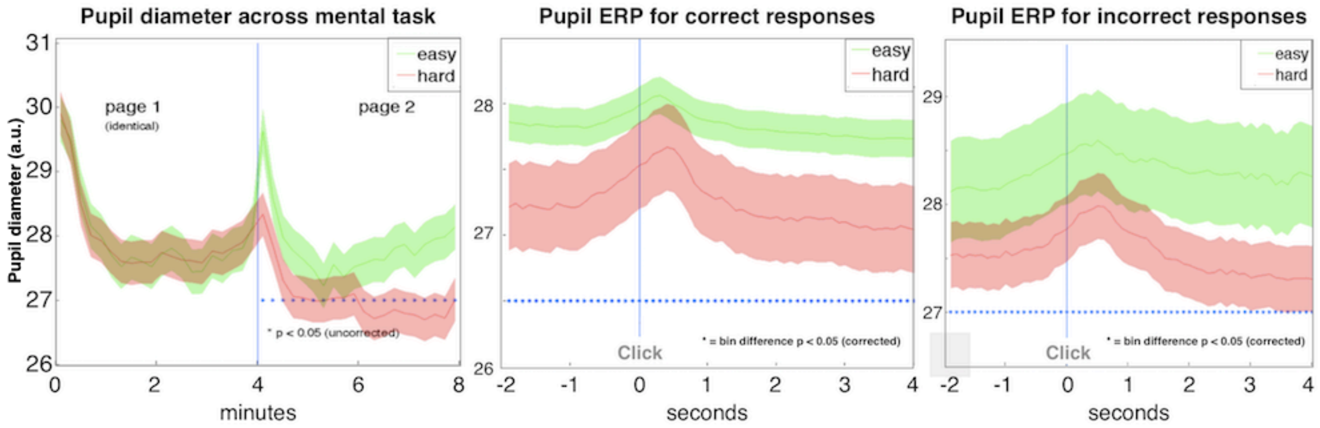

Present Study (n 77) **Behavioral Study Data (n 45)** **Replication with Pupil Data (n 44)**

* = Kruskal-Wallis $p < 0.05$

3.2 Pupil diameter is smaller during difficult mental task

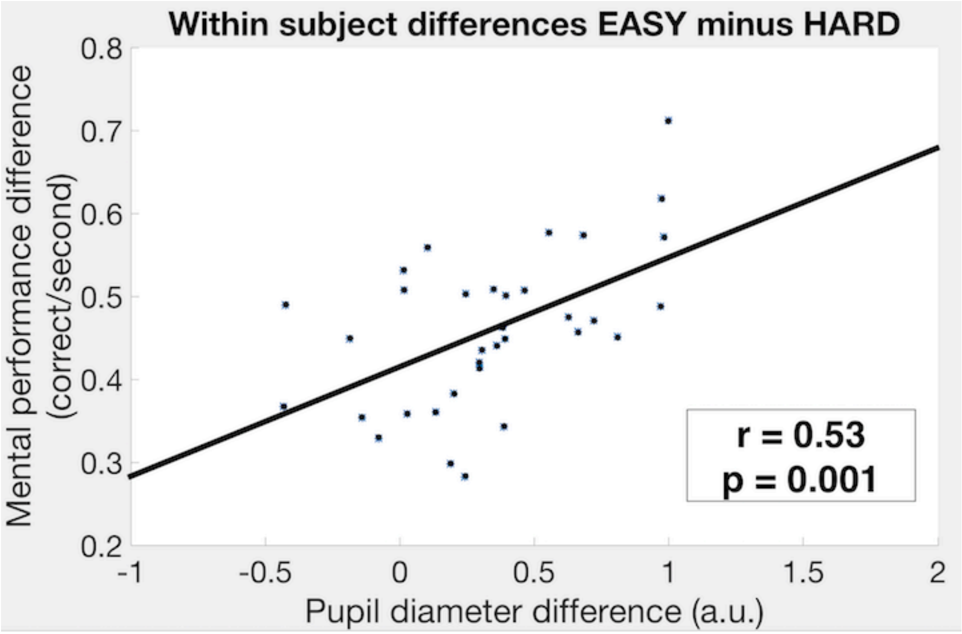
Pupil diameter was significantly smaller during the difficult mental effort task. This reduction in mean pupil size was not due to differences in pupil event related potentials (pERP) associated with finding target

“E’s”. The significant reduction in pupil is still present in pERP’s with a freely varying baseline.



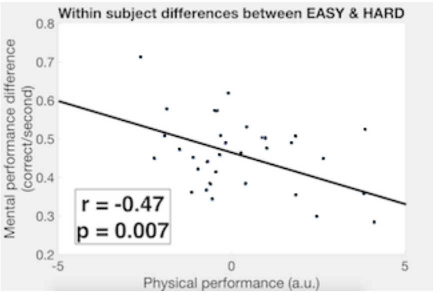
3.3 Pupil diameter during mental task correlates with mental performance

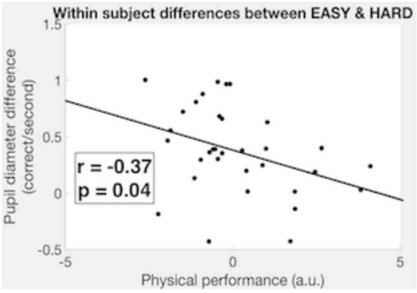
A reduced pupil diameter on a task designed to produce increases in cognitive demands was an unexpected result. Increases in mental workload are generally associated with increased arousal, resulting in increases in pupil size. We then investigated if decreased pupil size was representing systematic task disengagement by subjects. To do this we quantified performance on each page of the mental effort task as a mean reward rate (response accuracy/inter-response interval) as correct response per second. This measure captures the tradeoff between speed and accuracy to quantify total mental performance. For example, 80% accuracy with correct clicks every 2 second would yield a high score of 0.4 (0.8/2). High accuracy with long response times or low accuracy with fast response times produce middling scores (1.0/10 = 0.1 or 0.2/2 = 0.1). Individualized difference scores were created by subtracting page 2 (difficult) from page 1 (control). These scores were averaged across both difficult trials and then correlated with average pupil size difference scores calculated in the same way. It was found that *higher* relative mental performance correlated with *increased* relative pupil diameter. This correlation suggests that larger pupil size reflects task engagement. It should be noted that this relationship was only present in the condition when the hard mental tasks were completed in blocks 1 and 2 and easy in blocks 3 and 4.

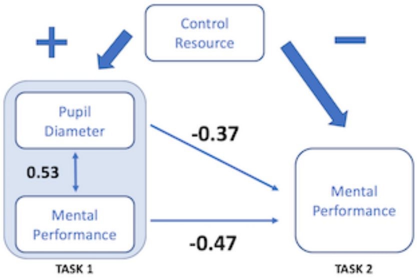


3.4 Pupil diameter and performance on difficult *mental* task correlates with *physical* performance

Relative pupil diameter and mental performance on difficult trials were then separately correlated with subsequent physical performance for each subject averaged across both difficult trials. It was found that lower relative mental performance (dM) predicted greater subsequent physical performance. This finding suggests that the performance changes observed after difficult mental tasks (the depletion effect) is mediated not only by categorical task difficulty, but also by individual performance on the difficult task (i.e., greater mental performance leads to greater depletion effect). Similarly, smaller relative pupil size (dP) predicted higher subsequent physical performance (T). This is consistent with an interpretation that increases in pupil size are indicative of increased task engagement with an associated increase in cognitive performance. Multiple regression analysis (T ~ dP + dM) revealed that the majority of the total variance is explained by mental performance difference alone.







Overall these findings suggest that the inconsistent results reported in the self-control depletion literature may be due in part to the failure to control for individual subject engagement on the depleting task. Using experimental paradigms that allow for quantification of individual performance on the depleting task combined with pupillometry (an increasingly affordable and non-invasive psychometric tool) offer a method for controlling for task engagement.

4 References

1. Zenon A, Sidibe M, Olivier E.. Journal of Neuroscience. 2015;35: 8737–8744.
2. Wickens CD. Theoretical Issues in Ergonomics Science. 2002;3: 159–177.
3. Kahneman D. Attention and effort. Prentice Hall; 1973.
4. Christie ST, Thomas Christie S, Schrater P. Front Neurosci. 2015;9.
5. Gailliot MT, Baumeister RF, et al. PsycEXTRA Dataset.
6. Muraven M, Tice DM, Baumeister RF.. J Pers Soc Psychol. 1998;74: 774–789.
7. Baumeister RF, Bratslavsky E, Muraven M, Tice DM. J Pers Soc Psychol. 1998;74: 1252–1265.
8. Hagger MS, Wood C, Stiff C, Chatzisarantis NLD. Psychol Bull. 2010;136: 495–525.
9. Inzlicht M, Schmeichel BJ. Perspect Psychol Sci. 2012;7: 450–463.
10. Hagger MS, Chatzisarantis NLD. Perspect Psychol Sci. 2016;11: 546–573.
11. Tingley D, Yamamoto T, Hirose K, Keele L, Imai K. J Stat Softw. 2014;59.

Attend, Adapt and Transfer: Attentive Deep Architecture for Adaptive Transfer from multiple sources in the same domain

Janarthanan Rajendran *
University of Michigan
`rjana@umich.edu`

Aravind S. Lakshminarayanan *
Indian Institute of Technology Madras
`aravindsrinivas@gmail.com`

Mitesh M. Khapra
Indian Institute of Technology Madras
`miteshk@cse.iitm.ac.in`

Prasanna P
McGill University
`prasanna.p@cs.mcgill.ca`

Balaraman Ravindran
Indian Institute of Technology Madras
`ravi@cse.iitm.ac.in`

Abstract

Transferring *knowledge* from prior source tasks in solving a new target task can be useful in several learning applications. The application of transfer poses two serious challenges which have not been adequately addressed. First, the agent should be able to avoid negative transfer, which happens when the transfer hampers or slows down the learning instead of helping it. Second, the agent should be able to selectively transfer, which is the ability to select and transfer from different and multiple source tasks for different parts of the state space of the target task. We propose A2T [1] (Attend, Adapt and Transfer), an attentive deep architecture which adapts and transfers from these source tasks. Our model is generic enough to effect transfer of either policies or value functions. Empirical evaluations on different learning algorithms show that A2T is an effective architecture for transfer by being able to avoid negative transfer while transferring selectively from multiple source tasks in the same domain.

Keywords: Reinforcement learning, Transfer Learning, Attention, Deep Reinforcement learning

Acknowledgements

Thanks to the anonymous reviewers of ICLR 2017 who provided thoughtful remarks and helped us revise our work. We would also like to thank Sherjil Ozair, John Schulman, Yoshua Bengio, Sarath Chandar, Caglar Gulchere and Charu Chauhan for useful feedback about the work.

---

[*] Authors contributed equally

[1] Published as a conference paper at ICLR 2017 `https://openreview.net/forum?id=Sy6iJDqlx&noteId=Sy6iJDqlx`

1 Introduction

One of the goals of Artificial Intelligence (AI) is to build autonomous agents that can learn and adapt to new environments. Reinforcement Learning (RL) is a key technique for achieving such adaptability. The goal of RL algorithms is to learn an optimal policy for choosing actions that maximize some notion of long term performance. Transferring knowledge gained from tasks solved earlier to solve a new target task can help, either in terms of speeding up the learning process or in terms of achieving a better solution, among other performance measures. When applied to RL, transfer could be accomplished in many ways ([4],[12],[6],[5] see [13, 14] for a very good survey of the field). One could use the value function from the source task as an initial estimate in the target task to cut down exploration [11]. Alternatively one could use policies from the source task(s) in the target task [1, 8, 7, 3]. The two recent works that are very relevant to the proposed architecture are [9] and [10].

While transfer in RL has been much explored, there are two crucial issues that have not been adequately addressed in the literature. The first is *negative transfer*, which occurs when the transfer results in a performance that is worse when compared to learning from scratch in the target task. This severely limits the applicability of many transfer techniques only to cases for which some measure of relatedness between source and target tasks can be guaranteed beforehand. This brings us to the second problem with transfer, which is the issue of identifying an appropriate source task from which to transfer. In some scenarios, different source tasks might be relevant and useful for different parts of the state space of the target task. As a real world analogy, consider multiple players (experts) who are good at different aspects of a game (say, tennis). For example, Player 1 is good at playing backhand shots while Player 2 is good at playing forehand shots. Consider the case of a new player (agent) who wants to learn tennis by selectively learning from these two experts. Our agent can transfer knowledge from Player 1 when required to play backhand shots and Player 2 for playing forehand shots. We call this *selective transfer*. Further, let us consider consider the situation that both Player 1 and Player 2 are bad at playing drop shots. Apart from the source tasks, we maintain a base network that learns from scratch on the target task. Our framework allows an agent to avoid transferring from both Players 1 and 2 while learning to play drop shots, and rather acquire the drop shot skill by learning to use the base network. The architecture is trained such that the base network uses not just the experience obtained through the usage of its solutions in the target task, but the overall experience acquired using the combined knowledge of the source tasks and itself. This enables the base network solutions to get closer to the behavior of the overall architecture (which uses the source task solutions as well). This makes it easier for the base network to assist the architecture to fine tune the useful source task solutions to suit the target task perfectly over time. The key contribution in the architecture is a *deep* ***attention*** *network*, that decides which solutions to attend to, for a given input state.

To this end, we propose A2T: Attend, Adapt and Transfer, an Attentive Deep Architecture for Adaptive Transfer, that avoids negative transfer while performing selective transfer from multiple source tasks in the same domain. A2T is a fairly generic framework that can be used to selectively transfer different skills available from different experts as appropriate to the situation. For instance, a household robot can appropriately use skills from different experts for different household chores. Further, A2T is generic enough to effect transfer of either action policies or action-value functions, as the case may be.

2 Proposed Architecture

Let there be N source tasks and let $K_1, K_2, \dots K_N$ be the solutions of these source tasks $1, \dots N$ respectively. Let K_T be the solution that we learn in the target task T. Source tasks refer to tasks that we have already learnt to perform and target task refers to the task that we are interested in learning now. These solutions could be for example policies or state-action values. Here the source tasks should be in the same domain as the target task, having the same state and action spaces. We propose a setting where K_T is learned as a function of $K_1, \dots, K_N, K_B$, where K_B is the solution of a base network which starts learning from scratch while acting on the target task.

$$K_T(s) = w_{N+1,s} K_B(s) + \sum_{i=1}^{N} w_{i,s} K_i(s) \tag{1}$$

$\sum_{i=1}^{N+1} w_{i,s} = 1, w_{i,s} \in [0,1]$ where, $w_{i,s}$ is the weight given to the ith solution at state s. The agent uses K_T to act in the target task. Figure 1 shows the proposed architecture. While the source task solutions $K_1, \dots, K_N$ remain fixed, the base network solutions are learnt and hence K_B can change over time. There is a central network which learns the weights ($w_{i,s}$, $i \in 1, 2, \dots, N+1$), given the input state s. We refer to this network as the *attention network*. The $[0,1]$ weights determine the attention each solution gets allowing the agent to selectively accept or reject the different solutions, depending on the input state. $(e_{1,s}, e_{2,s}, \dots, e_{N+1,s}) = f(s; \theta_a)$. Here, $f(s; \theta_a)$ is a deep neural network (attention network), which could consist of convolution layers and fully connected layers depending on the representation of input. It is parametrised by θ_a and takes as input a state s and outputs a vector of length $N+1$, which gives the attention scores for the $N+1$ solutions at state s. This score is then normalised using softmax to get the weights $w_{i,s}$. Working at the granularity of states allows the attention network to attend to different source tasks, for different parts of the state space of the target task, thus giving it the ability to perform selective transfer. For parts of the state space in the target task, where the source task solutions cause negative transfer or where the source task solutions are not relevant, the attention network learns to give high weight to the base network solution (which can be learnt and improved), thus avoiding negative transfer.

Depending on the feedback obtained from the environment upon following K_T, the attention network's parameters θ_a are updated to improve performance. The source task solutions, $K_1, \ldots, K_N$ remain fixed. Updating these source task's parameters would cause a significant amount of unlearning in the source tasks solutions and result in a weaker transfer, which we observed empirically. This also enables the use of source task solutions, as long as we have the outputs alone, irrespective of how and where they come from.

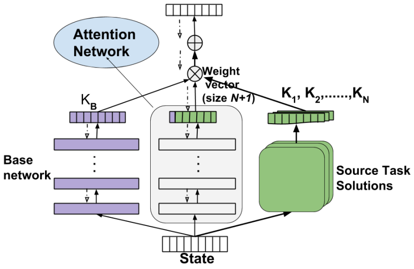


Figure 1: A2T architecture

Even though the agent follows K_T, we update the parameters of the base network that produces K_B, as if the action taken by the agent was based only on K_B. Due to this special way of updating K_B, apart from the experience got through the unique and individual contribution of K_B to K_T in parts of the state space where the source task solutions are not relevant, K_B also uses the valuable experience got by using K_T which uses the solutions of the source tasks as well. In practise, the source task solutions though useful, might need to be modified to suit perfectly for the target task. The base network also takes care of these modifications required to make the useful source task solutions perfect for the target task. The special way of training the base network assists the architecture in achieving this faster. To summarise, for a given state, A2T learns to *attend* to specific solutions and *adapts* this attention over different states, hence attaining useful *transfer*. A2T is general and can be used for transfer of solutions such as policy and value.

In the case of policy transfer, the solutions that we transfer are the source task policies, taking advantage of which, we learn a policy for the target task. Thus, we have $K_1, \ldots, K_N, K_B, K_T \leftarrow \pi_1, \ldots \pi_N, \pi_B, \pi_T$. Here π represents a stochastic policy, a probability distribution over all the actions. The agent acts in the target task, by sampling actions from the probability distribution π_T.

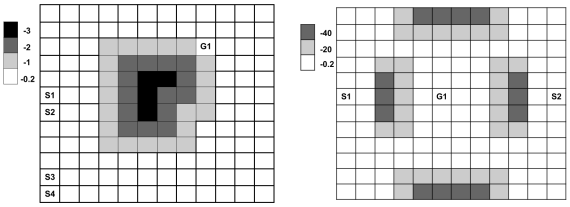
In the case of value transfer, the solutions being transferred are the source tasks' action-value functions, which we will call as Q functions. Thus, $K1, \ldots, K_N, K_B, K_T \leftarrow Q_1, \ldots, Q_N, Q_B, Q_T$. Let A represent the discrete action space for the tasks and $Q_i(s) = \{Q(s, a_j)\ \forall\ a_j \in A\}$. The agent acts by using Q_T in the target task, which is got as described in Eq.(1).

3 Experiments and Discussion

We evaluate the performance of A2T on policy transfer in two discrete versions of the standard puddle world (Figure 2a, 2b). Further, we evaluate the performance of our architecture on value transfer using the Arcade Learning Environment (ALE) platform [2]. The main goal of these experiments is to test the consistency of results with the algorithm motivation.

3.1 Ability to do Selective Transfer

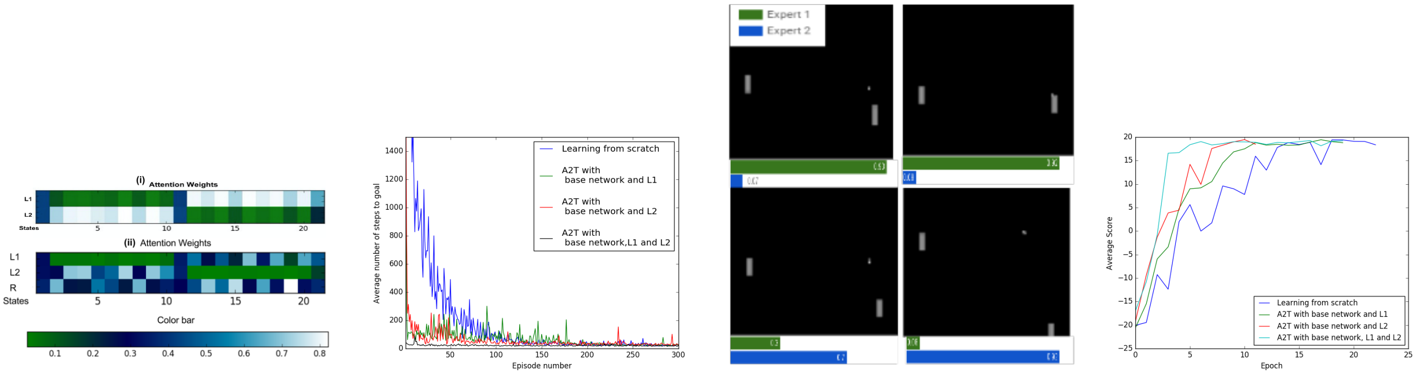
In this section, we consider the case when multiple partially favorable source tasks are available such that each of them can assist the learning process for different parts of the state space of the target task. The objective here is to first show the effectiveness of the attention network in learning to *focus* only on the source task relevant to the state the agent encounters while trying to complete the target task and then evaluating the full architecture with an additional randomly initialised base network.



(a) Puddle World 1 (b) Puddle World 2

Figure 2: Puddle worlds used for policy transfer experiments

This is illustrated for the Policy Transfer setting using the chain world of length 21. A is the left most cell, while B is the right most cell of the chain world. C lies in the center. Consider that the target task LT is to start in A or B with uniform probability and reach C in the least number of steps. $L1$ is the source task where the agent has learned to reach the left end (A) starting from the right end (B). In contrast, $L2$ is the source task where the agent has learned to reach the right end (B) starting from the left end (A).We train the agent to solve the task LT using REINFORCE given the policies learned for $L1$ and $L2$. Figure 3a (i) shows the weights given by the attention network to the two source task policies for different parts of the state space at the end of learning. We observe that the attention network has learned to ignore $L1$, and $L2$ for the left, and right half of the state space of the target task, respectively.



(a) The weights given by the attention network. Selective transfer in REINFORCE

(b) Selective transfer in Actor-Critic

(c) Visualisation of the attention weights in the Selective Transfer with Attention Network experiment:

(d) Selective Value Transfer.

Figure 3: Evaluation of A2T in doing selective transfer in different domains.

Next, we add base network and evaluate the full architecture on this task. Figure 3a (ii) shows the weights given by the attention network to the different source policies for different parts of the state space at the end of learning. We observe that the attention network has learned to ignore $L1$, and $L2$ for the left, and right half of the state space of the target task, respectively. As the base network replicates π_T over time, it has a high weight throughout the state space of the target task. We also evaluate our architecture in a relatively more complex puddle world shown in Figure 2b. In this case, $L1$ is the task of moving from $S1$ to $G1$, and $L2$ is the task of moving from $S2$ to $G1$. In the target task LT, the agent has to learn to move to $G1$ starting from either $S1$ or $S2$ chosen with uniform probability. We learn the task LT using Actor-Critic method where A2T is the actor. Figure 3b shows the performance results. We observe that actor-critic using A2T is able to use the policies learned for $L1$, and $L2$ and performs better than a network learning from scratch without any knowledge of source tasks.

We do a similar evaluation of the attention network, followed by our full architecture for value transfer as well. We try to simulate players with different weakness' in Pong. The trick is to blur the part of the screen where we want to force the agent to be weak at returning the ball. We construct two partially helpful source task experts $L1$ and $L2$. $L1$ is constructed by training a DQN on Pong with the upper quadrant (the agent's side) blurred, while $L2$ is constructed by training a DQN with the lower quadrant (the agent's side) blurred. The source task experts $L1$ and $L2$ scored an average of **9.2** and **8** respectively on Pong game play with black background. With an attention network to suitably weigh the value functions of $L1$ and $L2$, an average performance of **17.2** was recorded just after a single epoch (250,000 frames) of training. (The score in Pong is in the range of $[-21, 21]$). This clearly shows that the attention mechanism has learned to take advantage of the experts adaptively. Fig. 3c shows a visualisation of the attention weights for the same. We see that in the first two snapshots, the ball is in the lower quadrant and as expected, the attention is high on Expert-1, while in the third and fourth snapshots, as the ball bounces back into the upper quadrant, the attention increases on Expert-2.

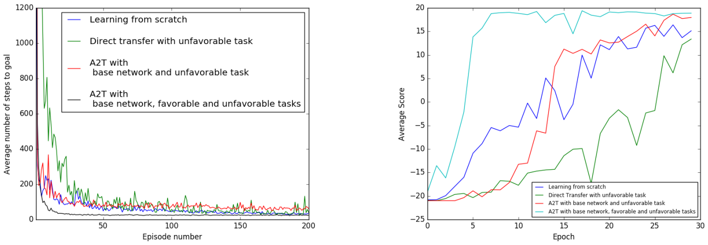
We then evaluate our full architecture (A2T) in this setting, i.e with an addition of DQN learning from scratch (base network) to the above setting. The architecture can take advantage of the knowledge of the source task experts selectively early on during the training while using the expertise of the base network wherever required, to perform well on the target task. Figure 3d summarizes the results, where it is clear that learning with both the partially useful experts is better than learning with only one of them which in turn is better than learning from scratch without any additional knowledge.

3.2 Ability to Avoid Negative Transfer and Ability to Transfer from Favorable Task

We first consider the case when only one learned source task is available such that its solution K_1 (policy or value) can hamper the learning process of the new target task. We refer to such a source task as an unfavorable source task. In such a scenario, the attention network shown in Figure 1 should learn to assign a very low weight (ignore) to K_1 . We also consider a modification of this setting by adding another source task whose solution K_2 is favorable to the target task. We now define an experiment using the puddle world from Figure 2a for policy transfer. The target task in our experiment is to maximize the return in reaching the goal state $G1$ starting from any one of the states $S1, S2, S3, S4$. We artificially construct an unfavorable source task by first learning to solve the above task and then negating the weights of the topmost layer of the actor network. We then add a favorable task to the above setting. We artificially construct a favorable source task simply by learning to solve the target task and using the learned actor network. Figure 4a shows the results. The target task for the value transfer experiment is to reach expert level performance on Pong. **Inverse-Pong**:

A DQN on Pong trained with negated reward functions, that is with $R'(s,a) = -R(s,a)$ where $R(s,a)$ is the reward provided by the ALE emulator for choosing action a at state s is used as the unfavorable task.

We artificially construct a favorable source task by learning a DQN to achieve expertise on the target task (Pong) and use the learned network. Figure 4b compares the performance of the various scenarios when the unfavorable source task is Inverse-Pong.



(a) Avoiding negative transfer and transferring policy from a favorable task(lower the better).

(b) Avoiding negative transfer(Pong) and transferring from a favorable task

Figure 4: Avoiding negative transfer and transferring value from a favorable task(higher the better).

From all the above results, we can clearly see that A2T does not get hampered by the unfavorable source task by learning to ignore the same and performs competitively with just a randomly initialized learning on the target task without any expert available. Secondly, in the presence of an additional source task that is favorable, A2T learns to transfer useful knowledge from the same while ignoring the unfavorable task, thereby reaching expertise on the target task much faster than the other scenarios.

4 Conclusion and Future work

In this paper we present a very general deep neural network architecture, A2T, for transfer learning that avoids negative transfer while enabling selective transfer from multiple source tasks in the same domain. We show simple ways of using A2T for policy transfer and value transfer. We empirically evaluate its performance with different algorithms, using simulated worlds and games, and show that it indeed achieves its stated goals. Apart from transferring task solutions, A2T can also be used for transferring other useful knowledge such as the model of the world. While in this work we focused on transfer between tasks that share the same state and action spaces and are in the same domain, the use of deep networks opens up the possibility of going beyond this setting. For example, a deep neural network can be used to learn common representations [9] for multiple tasks thereby enabling transfer between related tasks that could possibly have different state-action spaces. A hierarchical attention over the lower level filters across source task networks while learning the filters for the target task network is another natural extension to transfer across tasks with different state-action spaces. The setup from Progressive Neural Networks [10] could be borrowed for the filter transfer, while the A2T setup can be retained for the policy/value transfer. Exploring this setting for continuous control tasks so as to transfer from modular controllers as well avoid negative transfer is also a potential direction for future research.

References

[1] Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *In Proceedings of International Conference on Machine Learning*, volume 97, 1997.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *arXiv preprint arXiv:1207.4708*, 2012.

[3] Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 316–324, 2014.

[4] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727. ACM, 2006.

[5] George Konidaris, Ilya Scheidwasser, and Andrew G Barto. Transfer in reinforcement learning via shared features. *The Journal of Machine Learning Research*, 13(1):1333–1371, 2012.

[6] Alessandro Lazaric and Marcello Restelli. Transfer from multiple mdps. In *Advances in Neural Information Processing Systems*, pages 1746–1754, 2011.

[7] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71. ACM, 2004.

[8] Scott Niekum, Sachin Chitta, Andrew G Barto, Bhaskara Marthi, and Sarah Osentoski. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9, 2013.

[9] Emilio Parisotto, Jimmy Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *CoRR*, abs/1511.06342, 2015.

[10] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

[11] Jonathan Sorg and Satinder Singh. Transfer via soft homomorphisms. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 741–748. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[12] Erik Talvitie and Satinder Singh. An experts algorithm for transfer learning. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1065–1070. Morgan Kaufmann Publishers Inc., 2007.

[13] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

[14] Matthew E Taylor and Peter Stone. An introduction to intertask transfer for reinforcement learning. *AI Magazine*, 32(1):15, 2011.

Decision-Making with Non-Markovian Rewards: From LTL to automata-based reward shaping

Alberto Camacho
Department of Computer Science
University of Toronto
Toronto, Canada
`acamacho@cs.toronto.edu`

Oscar Chen[*]
Department of Engineering
University of Cambridge
Cambridge, UK
`ozhc2@cam.ac.uk`

Scott Sanner
Department of Mechanical & Industrial Engineering
University of Toronto
Toronto, Canada
`ssanner@mie.utoronto.ca`

Sheila A. McIlraith
Department of Computer Science
University of Toronto
Toronto, Canada
`sheila@cs.toronto.edu`

Abstract

In many decision-making settings, reward is acquired in response to some complex behaviour that an agent realizes over time. An autonomous taxi may receive reward for picking up a passenger and subsequently delivering them to their destination. An assistive robot may receive reward for ensuring a person in their care takes their medication once daily soon after eating. Such reward is acquired by an agent in response to following a path – a sequence of states that collectively capture the reward-worthy behaviour. Reward of this sort is referred to as non-Markovian reward because it is predicated on state history rather than current state. Our concern in this paper is with both the specification and effective exploitation of non-Markovian reward in the context of Markov Decision Processes (MDPs). State-of-the-art UCT-based planners struggle with non-Markovian rewards because of their weak guidance and relatively myopic lookahead. Here we specify non-Markovian reward-worthy behaviour in Linear Temporal Logic. We translate these behaviours to corresponding deterministic finite state automata whose accepting conditions signify satisfaction of the reward-worthy behaviour. These automata accepting conditions form the basis of Markovian rewards that can be solved by off-the-shelf MDP planners, while crucially preserving policy optimality guarantees. We then explore the use of reward shaping to automatically transform these automata-based rewards into reshaped rewards that better guide search. We augmented benchmark MDP domains with non-Markovian rewards and evaluated our technique using PROST, a state-of-the-art heuristic and UCT-based MDP planner. Our experiments demonstrate significantly improved performance achieved by the exploitation of our techniques. The work presented here reflects the use of Linear Temporal Logic to specify non-Markovian reward, but our approach will work for any formal language for which there is a corresponding automata representation.

Keywords: Markov Decision Processes, Non-Markovian Rewards, LTL, Monte Carlo Search, Reward Shaping

Acknowledgements

The authors gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC).

---

[*] This work was performed while the second author was affiliated with the University of Toronto.

1 Introduction

In Markov Decision Processes agents typically receive positive or negative reward in response to their current state. Nevertheless, agents may also realize reward in response to more complex behaviour that is reflected over a sequence of states. An autonomous taxi may receive reward for picking up a passenger and subsequently delivering them to their destination. Similarly, a personal robot getting ice cubes from the freezer is rewarded for opening the freezer, removing the ice cubes, *and* closing the freezer soon after. Such reward is commonly referred to as non-Markovian reward because it is predicated on the state history rather than solely on the current state. Our concern in this paper is with both the specification and effective exploitation of non-Markovian reward in Markov Decision Processes (MDPs). Here we use Linear Temporal Logic to specify non-Markovian rewards. Notwithstanding, our approach is applicable to other formal languages for which there exist corresponding automata representations.

Current state-of-the-art MDP planners are based on heuristic search and variants of UCT techniques [11]. UCT policies tend to make greedy and myopic decisions. As such, these planners struggle with non-Markovian rewards since there is little guidance for their relatively myopic lookahead. The impact of this myopic guidance can be seen in state-of-the-art MDP planner PROST [10], a UCT-based planner that generates high-quality solutions for moderately sized MDPs, but whose performance suffers in large problems that require significant lookahead.

In this paper we explore transformation of the reward function through reward shaping [12] as a means of mitigating for the myopic lookahead of UCT-based methods. To this end, we propose an approach to solving non-Markovian Reward Decision Problems (NMRDPs) by transforming our reward-worthy non-Markovian behaviour into corresponding deterministic finite state automata. The accepting conditions of these automata signify satisfaction of the reward-inducing behaviour in a manner that is solvable with off-the-shelf MDP planners, crucially preserving optimality guarantees. Moreover, we use reward shaping with these automata-based reward encodings in order to induce non-sparse, myopic-friendly rewards. This helps guide the accrual of non-Markovian reward. We evaluate our approach to solving NMRDPs via experimentation with off-the-shelf state-of-the-art heuristic and UCT-based MDP planners. Experiments with a set of International Probabilistic Planning Competition (IPPC) domains augmented with non-Markovian rewards show significantly improved performance using our automata representation together with reward shaping.

2 Background

2.1 Model-based Decision Making

Markov Decision Processes: *Markov Decision Processes* (MDPs) [14] are popular models for decision-theoretic planning problems [5]. An MDP is a tuple $M = \langle S, A, P, R, T, \gamma, s_0 \rangle$, where: S is a finite set of states; A is a finite set of actions; $P_a(s, s')$ is the probability of reaching the state $s' \in S$ after applying action a in state $s \in S$; $R : S \times A \times S \to \mathbb{R}$ is the reward function (sometimes $R : S \times A \to \mathbb{R}$); $T \in \mathbb{N}$ is the horizon; $\gamma \in (0, 1]$ is the discount factor; and $s_0 \in S$ is the *initial state* of the MDP. Solutions to an MDP are a sequence of step-dependent *policies* $\Pi = (\pi_0, \ldots, \pi_{T-1})$ that map states $s \in S$ at step k ($0 \leq k < T$) to actions $\pi_k(s) \in A$. The *value* of a policy Π in state s at step k, $V_{\Pi,k}(s)$, is the expected discounted cumulative reward over the horizon $T - k$ following Π. Formally, $V_{\Pi,k}(s) = \mathbb{E}_{\Pi}\{\Sigma_{i=k}^{T-1} \gamma^i R_i\}$, where R_t denotes the immediate reward obtained at step i if the agent follows policy Π from s. An optimal policy sequence Π^* for an MDP over horizon T with initial state s_0 satisfies $\Pi^* = \text{argmax}_{\Pi} V_{\Pi,0}(s_0)$.

MDPs are commonly described using factored representations of the states and dynamics. In particular, RDDL [15] is a modelling language that allows for a lifted, compact representation of factored MDPs. The current state-of-the-art solution method for MDPs specified in RDDL is PROST [10], a Monte Carlo sampling algorithm based on UCT and heuristic search. Whereas PROST generates good-quality solutions for moderate-sized MDPs, its performance suffers in large problems that require significant look-ahead. In such cases, the Monte-Carlo roll-outs cannot capture the structures inherent in the problem, leading to myopic search behavior.

Non-Markovian Reward Decision Processes: NMRDPs (e.g., [1, 2, 16]) generalize the MDP model by allowing reward functions to range over the history of visited states. In contrast to MDPs, the domain of the reward function R ranges over the set of finite state sequences drawn from S, denoted $S^\star$. As in conventional MDPs, optimal solutions maximize the expected discounted cumulative reward.

2.2 Finite Linear Temporal Logic and Deterministic Finite-State Automata

Linear Temporal Logic (LTL) is a compelling language for expressing temporal properties over (infinite) sequences of states, balancing expressiveness with ease of use. LTL was initially developed to express safety and liveness properties for program verification [13] and has been used for a myriad of applications including the specification of temporally extended goals and preferences in planning (e.g., [4]). Here we specify non-Markovian rewards using LTL$_\mathsf{f}$, a variant of LTL interpreted over finite traces. Our specification language is similar to $FLTL, a finitely interpreted future LTL used by

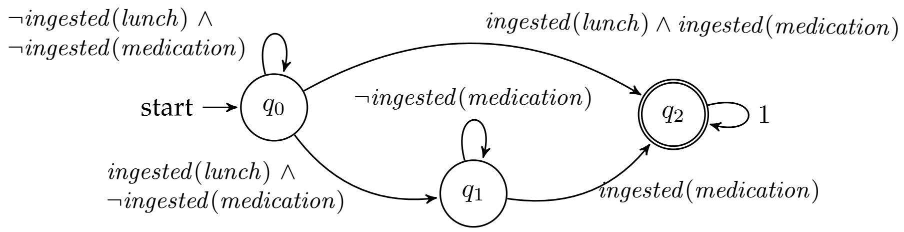

$\neg ingested(lunch) \wedge$
$\neg ingested(medication)$
$ingested(lunch) \wedge ingested(medication)$
start
q_0
$\neg ingested(medication)$
q_2
1
$ingested(lunch) \wedge$
$\neg ingested(medication)$
q_1
$ingested(medication)$

Figure 1: DFA corresponding to LTL$_\mathsf{f}$ formula $(\Diamond ingested(medication)) \wedge (\neg ingested(medication)\ \mathcal{U}\ ingested(lunch))$.

Thiébaux et al. (2006) to specify non-Markovian rewards. Bacchus et al. (1996, 1997) employed a finitely interpreted Past LTL (PLTL) to specify non-Markovian rewards.

The syntax of LTL$_\mathsf{f}$ includes the logical connectives ($\wedge$, $\vee$, $\neg$), unary modal operators *next* ($\bigcirc$), *weak next* ($\bullet$), and binary modal operator *until* ($\mathcal{U}$). Other operators, such as *always* ($\Box$) and *eventually* ($\Diamond$), can be defined in terms of these basic operators. The truth of an LTL$_\mathsf{f}$ formula φ is evaluated over *finite* sequences of states which – in the context of this paper – are the propositional states of an NMRDP or MDP (see [8] for details). We write $\pi \models \varphi$ when a sequence of states $\pi = s_0, \ldots, s_n$ satisfies φ.

By way of illustration, an assistive robot might accumulate reward by ensuring that its ward takes their medication daily and that they do so after eating lunch. Such behaviour might be expressed by LTL$_\mathsf{f}$ formula $(\Diamond ingested(medication)) \wedge (\neg ingested(medication)\ \mathcal{U}\ ingested(lunch))$.

2.2.1 LTL$_\mathsf{f}$ and Deterministic Finite-State Automata

Given LTL$_\mathsf{f}$ formula φ, one can construct a corresponding *Deterministic Finite-State Automaton* (DFA) A_φ that accepts a word π iff it satisfies φ (e.g. [9]). A DFA is a tuple $\langle Q, \Sigma, \delta, q_0, Q_{Fin}\rangle$, where Q is a finite set of states, Σ is the *alphabet* of the automaton, $\delta : Q \times \Sigma \to Q$ is a transition function, $q_0 \in Q$ is the initial state, and $Q_{Fin} \subseteq Q$ is a set of accepting states. The transition dynamics of an automaton is defined over *finite words*, or sequences $w = s_0, s_1, \ldots, s_n$ of elements in Σ. In the scope of this paper, Σ are the states of an MDP. At every stage i, the automaton makes a deterministic transition from state q_i to state $q_{i+1} = \delta(q_i, s_i)$. We say that M accepts w if $q_{n+1} \in Q_{Fin}$.

Fig. 1 shows a DFA corresponding to LTL$_\mathsf{f}$ formula $(\Diamond ingested(medication)) \wedge (\neg ingested(medication)\ \mathcal{U}\ ingested(lunch))$. Automaton states are represented by nodes, and transitions are represented by arcs. Transition labels describe the conditions that need to hold in a state to allow a particular transition. These are called *guards*. Finally, accepting states are depicted by double-ringed nodes. The word $\pi = \{\neg ingested(lunch), \neg ingested(medication); ingested(lunch), \neg ingested(medication); ingested(lunch), ingested(medication)\}$ induces one and only one run in the automaton, $\{q_0, q_0, q_1, q_2\}$. As this run finishes in an accepting state, it follows that π satisfies the LTL$_\mathsf{f}$ formula.

3 Problem: Solving NMRDPs with LTL$_\mathsf{f}$ rewards

As noted in Section 1, state-of-the-art MDP planners based on heuristic search and UCT struggle with non-Markovian rewards. In the rest of this paper we propose a novel method to effectively address this shortcoming. Following Bacchus et al. (1996), we specify rewards in an NMRDP as a *temporally extended reward function* (TERF). This TERF is realized by a set of reward behaviours, φ_i, specified in here in LTL$_\mathsf{f}$, together with a set of mappings to rewards r_i, denoted $\varphi_i : r_i$. Reward r_i is realized upon satisfaction of φ_i.

Returning to our previous example, we can define a TERF that gives a positive reward of 100 to agent behavior φ, satisfying $(\Diamond ingested(medication)) \wedge (\neg ingested(medication)\ \mathcal{U}\ ingested(lunch))$, written $\varphi : 100$. To only reward the first occurrence of the behaviour within a sequence of states, one could modify the above LTL$_\mathsf{f}$ formula as follows: $(\neg ingested(medication)\ \mathcal{U}\ (ingested(medication) \wedge \neg\bigcirc\top)) \wedge (\neg ingested(medication)\ \mathcal{U}\ ingested(lunch))$.

4 Approach: From LTL$_\mathsf{f}$ to automata-based reward shaping

To solve an NMRDP, M, we compile M with TERF R into an MDP M' with a Markovian reward R' that can be solved with a conventional off-the-shelf MDP planner. Our approach is realized in three steps: (i) for each $\varphi : r$ in the TERF, the LTL$_\mathsf{f}$ formula φ is transformed into a corresponding DFA A_φ; (ii) an MDP M' is constructed from M by augmenting state variables and transitions to reflect the state and progress of each A_φ towards its accepting condition. The Markovian reward function R' is associated with being in the accepting conditions of each A_φ, denoting satisfaction of reward-worthy behaviour φ; and (iii) M' is solved using an off-the-shelf MDP planner, thus obtaining a solution that can be converted, in a straightforward manner, into a solution to M. For the purposes of this paper, we limit our explication to finite-horizon NMRDPs. Notwithstanding, our approach can be extended to infinite-horizon NMRDPs.

Elaborating on step (ii), M' augments M with extra fluents and actions that integrate the dynamics of the DFAs within the MDP. The dynamics of M' expand *each* time step into three modes: **world**, **sync**, and **reward**. In **world** mode, an action from the NMRDP is applied. In **sync** mode, the automata states are synchronized according to the observed state. Intuitively, the automata states simulate the runs of the automata given the observed **world** state trajectories. The assignment of reward is delayed to **reward** mode and is performed upon satisfaction of each of the $\mathsf{LTL_f}$ reward formulae in the TERF . This is detected when an automaton reaches an accepting state. Solutions to the original NMRDP– that is, mappings from state trajectories into actions – can be obtained from solutions to the compiled MDP M' by simulating the state trajectories in M'. Our compilation preserves optimal solutions, as there is a bijection between state trajectories in M and M' that preserves the accumulated reward. The interested reader can find the technical details in [6, 7].

Theorem 1. *The automata-based compilation from NMRDPs into MDPs preserves optimal solutions.*

Returning to our assistive robot example with TERF defined by $\varphi : 100$, suppose the agent performs actions $\mathit{ingest}(\mathit{lunch})$ followed by $\mathit{ingest}(\mathit{medication})$, which induce the state trajectory (only relevant subset of state shown): $\pi = \{\neg \mathit{ingested}(\mathit{lunch}), \neg \mathit{ingested}(\mathit{medication}); \mathit{ingested}(\mathit{lunch}), \neg \mathit{ingested}(\mathit{medication}); \mathit{ingested}(\mathit{lunch}), \mathit{ingested}(\mathit{medication})\}$. The dynamics in the compiled MDP start by processing the initial state, and self-transitioning from the automaton state q_0 to itself. In **reward** mode, no reward is given. Then, in **world** mode the action $\mathit{ingest}(\mathit{lunch})$ is performed, leading to a state s_1 in which $\{\neg \mathit{ingested}(\mathit{medication}), \mathit{ingested}(\mathit{lunch})\}$ holds. The following **sync** mode synchronizes the automaton state to q_1, and so on until reaching world state s_2, where $\{\mathit{ingested}(\mathit{medication}), \mathit{ingested}(\mathit{lunch})\}$ holds. At this point, the automaton synchronizes to state q_3, that is accepting. In **reward** mode, a reward of 100 is given.

4.1 Improving Performance via Reward Shaping

The above approach to solving NMRDPs preserves optimality (cf. Theorem 1). Here we augment our approach with reward shaping in an effort to mitigate for the sparse reward inherent in our non-Markovian rewards, that aggravates the weak guidance and lookahead of state-of-the-art UCT-based MDP planners.

Reward shaping is a common technique in MDPs which aims to improve search by transforming the reward function. Such reward transformations have the form $R'(s, a, s') = R(s, a, s') + F(s, a, s')$, where R is the original reward function and F is a *shaping reward function*. The intuition behind reward shaping is that by increasing (resp. decreasing) the reward in states that lead to other high-value states or trajectories (resp. low-value states or trajectories), we can increase the effectiveness of search and the quality of solutions, while reducing search memory and run times. Unfortunately, reward shaping with an arbitrary $F(s, a, s')$ may lead to an optimal policy that is suboptimal w.r.t. the original unshaped reward. However, as noted by [12], if $F(s, a, s')$ is chosen from a restricted class of *potential-based* shaping functions defined as $F(s, a, s') = \gamma\phi(s') - \phi(s)$ (for some real-valued function ϕ), then this guarantees preservation of optimal and near-optimal policies with respect to the original unshaped MDP. Preservation of near-optimality is desirable since it provides guarantees for suboptimal solutions obtained by approximate methods.

Here we introduce a reward shaping technique that is, by construction, potential-based, and thus preserves optimal and near-optimal solutions. Given a particular automaton configuration in s, the idea is to decompose the potential $\phi(s)$ into sums of potentials for each of the automaton state variables, f_q, that together describe the current automaton configuration. Since many off-the-shelf MDP planners employ reward functions of the form $R(s, a)$, rather than the more general $R(s, a, s')$, we record the previous automaton configuration in corresponding state variables f_q^c, and delay attribution of reward to **reward** mode. Our corresponding shaping reward function is $F(s) = \gamma \sum_{f_q} \phi(f_q) - \sum_{f_q^c} \phi(f_q^c)$, for all f_q and f_q^c that hold in s. To preserve optimality in finite-horizon NMRDPs, partial rewards given to state trajectories that do not finish in accepting states of the TERF must be subtracted; full details are in [7].

Theorem 2. *Automata-based reward shaping preserves optimal, and near-optimal solutions.*

In the assistive robot example, we may want to provide some guidance by assigning potentials $\phi(q_0) = 0$, $\phi(q_1) = 50$, and $\phi(q_2) = 100$. Intuitively, these potentials assign positive reward for transitioning from q_0 to q_1, with the rationale that state trajectories that yield such transitions make progress towards achievement of an accepting state.

5 Empirical Evaluation

We performed preliminary experiments to assess the benefits of our automata-based reward transformations at guiding search for high-quality solutions. For this task, we selected two different benchmarks from the previous IPPC: *academic-advising*, and *wildfire*. We replaced the original rewards with non-Markovian rewards, and compiled the resulting NMRDPs into MDPs as described above. All problems were described in RDDL [15], and we used PROST [10] as the MDP planner. In *academic-advising* problems, p_m_n, the agent is rewarded for passing $m \cdot n$ courses, each one having m course

MDP Planner	Compilation	p-3-3	p-4-2	p-4-3	p-4-4
PROST UCT*(IDS)	MDP	30	30	0	0
PROST UCT*(DFS)	MDP	30	30	0	0
PROST IPPC-2014	MDP	2	30	0	0
PROST IPPC-2011	MDP	27	30	2	0
UCT	MDP	0	0	0	0
PROST UCT*(IDS)	MDP + RS	30	30	30	30
PROST UCT*(DFS)	MDP + RS	30	30	30	30
PROST IPPC-2014	MDP + RS	30	30	30	30
PROST IPPC-2011	MDP + RS	30	30	30	30
UCT (3 steps look ahead)	MDP + RS	29	30	29	30

Table 1: Number of runs (over 30 trials) that achieved the non-Markovian reward in the *academic-advising* problems. Compilations tested with and without reward shaping (RS).

MDP Planner	No RS	RS
PROST UCT*(IDS)	MLE	617
PROST UCT*(DFS)	MLE	627
PROST IPPC-2014	MLE	620
PROST IPPC-2011	MLE	637
UCT (3 steps look ahead)	423	527
no actions taken	263	263

Table 2: Average reward achieved (over 30 trials) in the MDP compilations of the *wildfire* problem, with and without reward shaping (RS). MLE indicates that PROST exceeded the memory limit of 512 MB.

prerequisites. In the *wildfire* problems, the agent is given reward to extinguish fire in a 3×3 grid, if performed no later than two time steps from its origination. The potentials used for reward shaping are naive, and distributed uniformly in the automaton states according to the distance to an accepting state. We observed that reward shaping can be an effective technique to provide guidance. In some of our tests, the quality of solutions improved drastically from zero average reward, to optimal policies even with simple UCT search (see Table 1). The *wildfire* problems are more probabilistically complex, and PROST easily ran out of memory (512 MB) in problems without reward shaping (see Table 2). On the other side, the guidance provided by reward shaping reduced the amount of memory needed by PROST, which found policies of increased quality than those obtained without reward shaping – even when we limited the memory usage of PROST.

6 Summary and Discussion

NMRDPs provide a powerful framework for modelling decision-making problems with behaviour-based rewards. In this paper we use $\mathsf{LTL_f}$ to specify rich non-Markovian rewards and present a technique for solving NMRDPs through a compilation to MDPs that can be solved with off-the-shelf MDP planners. Our approach integrates automata representations of the $\mathsf{LTL_f}$ formulae into the compiled MDP. We leverage reward shaping to help guide search, mitigating for the sparseness of non-Markovian rewards and the poor lookahead of some state-of-the-art UCT-based methods. Our experiments demonstrate that automata-based reward shaping is an effective method to enhance search and obtain solutions of superior quality. While non-Markovian rewards were specified here in $\mathsf{LTL_f}$, the proposed approach will work for rewards specified in any formal language for which there is a corresponding automata representation (e.g., [3]).

References

[1] F. Bacchus, C. Boutilier, and A. J. Grove. Rewarding behaviors. In *AAAI*, pages 1160–1167, 1996.

[2] F. Bacchus, C. Boutilier, and A. J. Grove. Structured solution methods for non-markovian decision processes. In *AAAI*, pages 112–117, 1997.

[3] J. A. Baier, C. Fritz, M. Bienvenu, and S. McIlraith. Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *AAAI*, pages 1509–1512, 2008.

[4] J. A. Baier, F. Bacchus, and S. A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173(5-6):593–618, 2009.

[5] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR*, 11:1–94, 1999.

[6] A. Camacho, O. Chen, S. Sanner, and S. A. McIlraith. Non-Markovian rewards expressed in LTL: Guiding search via reward shaping. In *SoCS*, 2017. To appear.

[7] A. Camacho, O. Chen, S. Sanner, and S. A. McIlraith. Decision-making with non-markovian rewards: Guiding search via automata-based reward shaping. Technical Report CSRG-632, Department of Computer Science, University of Toronto, 2017.

[8] G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860, 2013.

[9] G. De Giacomo and M. Y. Vardi. Synthesis for LTL and LDL on finite traces. In *IJCAI*, pages 1558–1564, 2015.

[10] T. Keller and P. Eyerich. PROST: probabilistic planning based on UCT. In *ICAPS*, 2012.

[11] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

[12] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations : Theory and application to reward shaping. In *ICML*, volume 3, pages 278–287, 1999. doi: 10.1.1.48.345.

[13] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[14] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994. ISBN 0471619779. doi: 10.1080/00401706.1995.10484354.

[15] S. Sanner. Relational dynamic influence diagram language (RDDL): Language description. http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf, 2010.

[16] S. Thiébaux, C. Gretton, J. K. Slaney, D. Price, F. Kabanza, et al. Decision-theoretic planning with non-markovian rewards. *JAIR*, 25:17–74, 2006.

Sample-Efficient Reinforcement Learning for Robot to Human Handover Tasks

Trevor Barron, Heni Ben Amor
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, AZ 85281
tpbarron@asu.edu, hbenamor@asu.edu

Abstract

While significant advancements have been made recently in the field of reinforcement learning, relatively little work has been devoted to reinforcement learning in a human context. Learning in the context of a human adds a variety of additional constraints that make the problem more difficult including an increased importance on sample efficiency and the inherent unpredictability of the human counterpart.

In this work we used the Sparse Latent Space Policy Search algorithm and a linear-Gaussian trajectory approximator with the objective of learning optimized, understandable trajectories for object handovers between a robot and a human with very high sample-efficiency. We present an analysis of various learning scenarios and provide results for each.

Keywords: Trajectory optimization, human-robot interaction, dimensionality reduction

Acknowledgements

This work was supported by Toyota InfoTechnology Center.

1 Introduction

Reinforcement learning (RL) algorithms have been shown to solve highly challenging tasks. Harnessing these capabilities for human-assistive technologies will lead to important breakthroughs in domestic and healthcare robotics.

For example, a robot assistant may learn how to seamlessly handover objects to human partners with specific personal preferences, limited postural range, or physical disabilities. Fig. 1 shows a robot assistant offering an apple to a person with an arm fracture. By finding an individualized solution for the task, the robot could increase the likelihood of success and improve quality of life for the human. Unfortunately, standard reinforcement learning methods are ill-suited for human-in-the-loop learning scenarios due to their inherent sample complexity required to reach a reasonable solution. Both human and robot would have to jointly execute a task for long periods of time before finding a reasonable solution.

Human-centered applications of RL therefore require sample-efficient methods that can learn with a limited budget of trials, e.g., less than 100 trials. In this paper we discuss a sample-efficient RL method that is particularly well suited for human-in-the-loop learning. The rationale underlying our algorithm is that policy parameters are often correlated. Individual joints can be grouped into synergies. A combination of a small number of synergies, can lead to a large range of different possible movements. In humans, such synergies reduce the dimensionality of the control task and, in turn, reduce the cognitive effort during learning and execution [1].

Leveraging this insight, we factorize robot control policies into a matrix of synergies, as well as a low-dimensional control policy. The synergies and control policies are updated within a reinforcement learning loop thus exhausting the information provided by each trial. The result is a sample-efficient policy search method for motor skill tasks that involve the coordination of multiple joints. In human-robot collaboration experiments, the robot autonomously learned optimal handover strategies within 30-40min.

We evaluate the approach in different robot to human handover settings involving a static and mobile robot, different postural ranges of the human partner, along with three different cost functions that we propose for reinforcement learning in a human context. We emphasize that the setup is not specific to handover tasks and that the paradigm should generalize to additional scenarios.

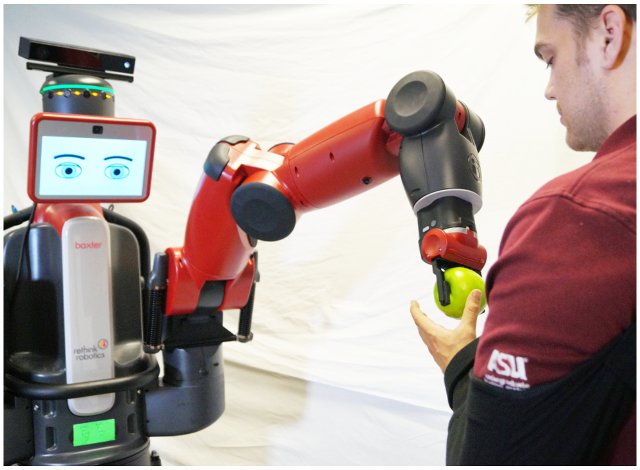


Figure 1: A Baxter robot learning to handover an apple to a person with an arm fracture. Adaptation is performed through sample-efficient reinforcement learning.

2 Methodology

2.1 Sparse Latent Space Policy Search

Policy search methods try to find an optimal policy for an agent which acts in an uncertain world with an unknown world model. Such methods use a parametrized policy represented by $\pi_\lambda(\mathbf{a}_t|\mathbf{s}_t,t)$ with parameters λ. The goal is to identify the values for λ that maximize the expected return of the policy:

$$J(\lambda) = \int_{\mathbb{T}} p_\lambda(\tau)R(\tau)d\tau \tag{1}$$

where the expectation integrates over all possible trajectories τ in the set $\mathbb{T}$ of time steps. Each trajectory $\tau = [\mathbf{s}_{1:T}, \mathbf{a}_{1:T}]$ is the result of repeatedly applying policy π_λ in all states $\mathbf{s}$ the agent encounters.

Our goal is to create a specific policy search algorithm for collaborative tasks that converges on an optimal policy with less than 100 executions. This goal can be achieved by reducing the dimensionality of the reinforcement learning process. Using latent variable estimation techniques, Luck, et. al. [2] derive a policy search variant that uncovers the low-dimensional manifold of solutions during the search process. To this end, we factorize the action at time step t according to:

$$\mathbf{a}_t^{(m)} = \left(\mathbf{W}^{(m)}\mathbf{Z}_t + \mathbf{M}^{(m)} + \mathbf{E}_t^{(m)}\right)\phi(\mathbf{s}_t,t), \tag{2}$$

where $m \in [1,2,\ldots,M]$ defines the number of the group and $\mathbf{a}_t^{(m)} \in \mathbb{R}^{D_m\times 1}$ the D dimensions of the action vector for a specific time step t. The vector $\mathbf{a}_t$ is a linear projection of the feature vector $\phi(\mathbf{s}_t,t) \in \mathbb{R}^{p\times 1}$. The mean matrix is given by $\mathbf{M} \in \mathbb{R}^{D\times p}$ and the exploration noise by the entries of $\mathbf{E} \in \mathbb{R}^{D\times p}$.

The above factorization can be seen as a low-dimensional policy that is projected into a higher-dimensional space through the projection matrix $\mathbf{W}$. This yields a class of policy search algorithms that simultaneously searches for a lower-dimensional policy, as well as the projection matrix $\mathbf{W}$ that embeds the low-dimensional latent signal into the high-dimensional space of robot control parameters. Using such a *latent space* approach for robot learning results in highly sample efficient policy search methods that can learn from a small number of executions in the real-world.

2.2 Policy Representation

We use a linear-Gaussian time-dependent trajectory representation similar to a Dynamic Motor Primitive (DMP) [3]. However, in contrast to DMPs, we do not employ a forcing function. A trajectory is represented by a set of weights and their respective basis functions which are spaced equally over the time of the trajectory.

In order to ensure invariance with respect to the position and orientation of the user, the linear-Gaussian policy is defined in the coordinate frame of the human partner (as opposed to the joint space of the robot). The trajectory can be transformed to robot space and then converted to joint space by using inverse kinematics. Defining the policy in the human frame permits a simpler generalization, as the trajectory is specified relative to human movements. Additionally, constraints on human posture can also be easily encoded in this space. Finally, representing policies in the coordinate space of the human permits training over fewer parameters than using joint coordinates.

Before running RL, imitation learning is used to derive an initial policy. We provided a few samples (ten in our experiments) defining a reasonable handover trajectory. Consequently, the trajectory representation is fit to these samples.

2.3 Reward Functions

Subsequently, we introduce various reward functions that are well-suited for human-robot interaction scenarios.

Optimizing Distance: Perhaps the most intuitive objective when learning a handover task is the distance between the robot's end-effector and the human's hand. This turned out to be a quite effective method but was susceptible to learning unintuitive trajectories since the only reward signal was the final distance. Accordingly, we try two additional methods to resolve the jerkiness in the trajectory while still optimizing for distance. Formally, the cost function we minimize is

$$D(W,H) = \frac{1}{2}\sum_i \|\boldsymbol{\psi}_i^T W - \boldsymbol{h}_i\|^2 e^{-(t-i)}. \tag{3}$$

Symbol	*Meaning*
t	Total number of time steps in a trajectory
d	Total number of degrees of freedom
b	Total number of basis functions. Also the number of weights per DOF.
Ψ	The set of Gaussian basis functions spaced over time in $\mathbb{R}^{b \times t}$
W	The weights on the basis functions that define a trajectory in $\mathbb{R}^{b \times d}$
H	A matrix representing the position of the human in each DOF over time in $\mathbb{R}^{t \times d}$

Table 1: Notation used to describe our cost functions

The notation used throughout the paper can be found in Tab. 1. We add an exponential scaling term that gives a higher weight to time steps close to the final position.

Optimizing Jerk: To combat some of the jerky, counterintuitive trajectories that can result from optimizing only for distance we also try optimizing over a combination of closeness to the human's hand and minimal trajectory jerk. This motivation behind this objective stems from an experimentally validated theory that humans make arm movements with minimal jerk trajectories [4]. The jerk cost for a trajectory is the sum of squared jerk over time averaged over degrees of freedom.

$$S(W) = \frac{1}{2d}\sum_i \left(\frac{\delta^3 \boldsymbol{\psi}_i^T W}{\delta t^3}\right)^2 \tag{4}$$

The total cost is then a linear combination of the jerk cost and the distance cost. We perform a grid search to find an approximate minimum of $C(W) = D(W,H) + \lambda S(W)$ and find an approximate solution at $\lambda = 0.5$.

3 EXPERIMENTS

We ran all of our experiments on the Rethink Robotics Baxter robot mounted on a Clearpath Robotics Ridgeback omnidirectional base. We use Kinect skeleton tracking to monitor the human position. We run several experiments analyzing various reward functions for human robot interaction along with eight different scenarios that validate our robot to human handover approach. First we describe the various objectives and then each scenario. Four human subjects performed each of the eight scenarios.

3.1 Human-In-The-Loop Experiments

We run a total of 32 experiments with four human subjects: with and without the mobile base, with the human sitting and standing, and with and without a sling simulating a lack of mobility. All of our experiments used eight iterations and 10 interactions per iteration. In all human experiments we optimize only for distance from the end-effector to the human's left hand. This results in 90 training interactions (20 in the first iteration).

4 Results

We achieve good convergence in all scenarios and often find a good solution in less than five iterations, or about 50 interactions. Table 2 shows specific results per task per subject. Due to noise in the Kinect skeleton tracking and variability in the human we cannot expect to reach a distance of zero. Averaged over all trials, we reach a final distance of approximately twenty centimeters, which accounting for these sources of error, is well within the range of a grasp target even for someone with a mobility impairment.

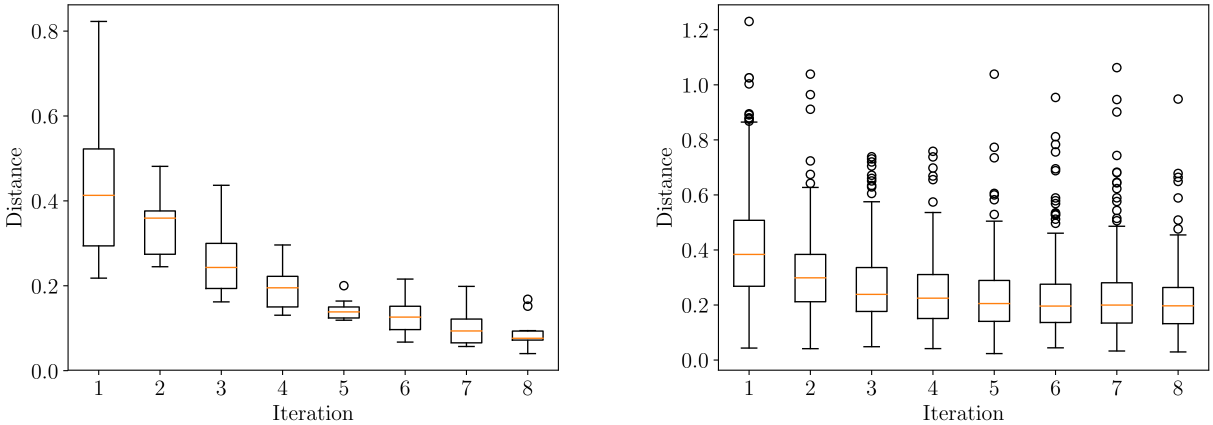


Figure 2: Distribution of distances calculated from a single experiment (left) and all performed experiments (right) at each iteration. We can see that even averaged over all experiments the distance to the user is shrinking.

Fig. 2 (left) shows the variance in distance from within a single RL experiment. During the initial trials, high exploration produces a range of different distance values. However, as learning progresses, the distance shrinks to below 10cm.

Scenario	*Subject #1*	*Subject #2*	*Subject #3*	*Subject #4*
Non-Mobile				
├─Standing				
│ ├─No-Sling	0.124	**0.063**	**0.040**	0.095
│ └─Sling	0.204	0.144	*0.205*	0.150
└─Sitting				
├─No-Sling	**0.092**	0.118	0.070	0.152
└─Sling	0.177	0.203	0.116	0.363
Mobile				
├─Standing				
│ ├─No-Sling	0.142	0.224	0.174	0.144
│ └─Sling	*0.342*	*0.313*	0.181	*0.374*
└─Sitting				
├─No-Sling	0.110	0.202	0.168	0.119
└─Sling	0.200	0.097	0.113	**0.054**

Table 2: Results per individual per task measured in meters between the endeffector and the human's hand. The best trial for each subject is bolded; the worst trial is italicized.

While we used only the distance cost in the optimization, the trend of the other costs throughout training is still interesting. We find that optimizing for distance still results in reasonably low jerk but appears to have little to no effect on correlation. During the first iteration exploration is high and results in very large jerk costs. After the first policy update, we achieve a similar jerk cost to the initial policy and it does not decrease further.

It is perhaps most instructive to examine when the algorithm performed very well or very poorly. Notice that three of the four subjects worst results were on tasks involving the mobile base. While a sample size of four is not sufficient to make a statistical conclusions we do believe the adding base movement (even only in one dimension) makes the task much more difficult. This relates to reward function design. Unless we hand engineer the reward to take into account both base movement and arm movement, an interaction could have excellent base movement but still receive a high cost due to poor arm movement. The opposite is also possible. Learning in this scenario requires that exploration early in training experience high value states, which becomes increasingly less likely as the dimensionality increases.

5 Subjective Observations

All of the human subjects in our experiments had prior experience working with robots. Since our emphasis in this work is on efficient learning in the context of a human we did not conduct a formal survey. Nevertheless we did gather some observations of the training experience. None of the subjects reported feeling unsafe during the process though all were familiar with the Baxter robot. Still, safety is a major concern when learning a human context. Our setup had no constraints that stopped the base from driving into the human or hitting the human mid trajectory. We did not experience any safety issues but these are important considerations. The most common observation was that the movement of the robot was unpredictable especially early during training with exploration is high.

6 Related Work

There is some existing work with ties to our own. DMPs are a very common means of trajectory representation in robotics and have been used in reinforcement learning applications but to our knowledge have not been used in a interactive context [5]. Likewise, imitation learning has become a common method of initializing policies, especially in robotics, to provide prior information and reduce training time [6]. The existing work in human-robot learning has largely been focused on learning distributions over trajectories using imitation learning. Ben Amor developed the Interaction Primitives framework, an extension of DMPs that also models the human movement and conditions the trajectory of the robot on the observation of the human [7],[8], [9]. Unlike an Interaction Primitive, our work only determines the trajectory based on the initial position of the human and does not condition the robot movement on additional observations of the human over time. This is the logical next step in our work. Finally, [10] also adapt handover trajectories to human limitations. Unlike our work, they also perform an estimation step over the rewards which are specified by the human.

7 Conclusion and Future Work

This work examines a sample efficient method for reinforcement learning in human-robot interaction, specifically handover tasks. While our approach does converge to good solutions, we feel the need to mention some significant limitations whose resolution we leave for future work. First, since each trajectory timestep is not conditioned on the state of the human, the current setup generalizes poorly with human movement. Second, since the algorithm adds random noise to the model weights for each interaction, the movement along the trajectory is not guaranteed to be time correlated. This often results in trajectories during exploration that are not intuitive to the human counterpart.

References

[1] M. Santello, M. Flanders, and J. Soechting, “Postural hand synergies for tool use,” *The Journal of Neuroscience*, vol. 18, no. 23, 1998.

[2] K. S. Luck, J. Pajarinen, E. Berger, V. Kyrki, and H. B. Amor, “Sparse latent space policy search.” in *AAAI*, 2016, pp. 1911–1918.

[3] S. Schaal, “Dynamic movement primitives-a framework for motor control in humans and humanoid robotics,” in *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.

[4] T. Flash and N. Hogan, “The coordination of arm movements: an experimentally confirmed mathematical model,” *Journal of neuroscience*, vol. 5, no. 7, pp. 1688–1703, 1985.

[5] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, “A survey on policy search for robotics,” *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[6] S. Schaal, “Is imitation learning the route to humanoid robots?” *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[7] H. B. Amor, D. Vogt, M. Ewerton, E. Berger, B. Jung, and J. Peters, “Learning responsive robot behavior by imitation,” in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 3257–3264.

[8] H. B. Amor, G. Neumann, S. Kamthe, O. Kroemer, and J. Peters, “Interaction primitives for human-robot cooperation tasks,” in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2831–2837.

[9] M. Ewerton, G. Neumann, R. Lioutikov, H. B. Amor, J. Peters, and G. Maeda, “Learning multiple collaborative tasks with a mixture of interaction primitives,” in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1535–1542.

[10] A. Kupcsik, D. Hsu, and W. S. Lee, “Learning dynamic robot-to-human object handover from human feedback,” *CoRR*, vol. abs/1603.06390, 2016. [Online]. Available: http://arxiv.org/abs/1603.06390

Robust Extensions to Policy Gradient Methods

Rishi Shah
The University of Texas at Austin
`rishi@cs.utexas.edu`

Jivko Sinapov
The University of Texas at Austin
`jsinapov@cs.utexas.edu`

Peter Stone
The University of Texas at Austin
`pstone@cs.utexas.edu`

Abstract

Reinforcement learning is a computational approach to learning from interaction. Specifically, it describes the framework for modelling an agent interacting with an environment as a Markov Decision Process (MDP). A class of techniques called *policy gradient methods* are used to solve MDPs by optimizing a parametrized policy via the use of stochastic gradient descent. The major goal of this paper is to use the tools of convex optimization to improve these methods.

First, we draw inspiration from LASSO and add ℓ_1 regularization to our policy gradient objective in order to induce sparse parameters. The motivation for doing so comes from the fact that many domains have noisy and irrelevant features that would benefit from sparsity.

Next, we introduce Mirrored PG, which applies mirror descent to policy gradient methods. Many mirror maps, such as p-norms, have been shown to handle noise particularly well, and furthermore, there exists a rich collection of feasible mirror maps. For this reason, we integrate mirror descent into the policy gradient framework.

Keywords: policy gradient, mirror descent, reinforcement learning

1 Related Work

ℓ_1 regularization has shown up in many value-function based methods; examples include LARS-TD [3] and Sparse Q-Learning [4]. Motivated by the success of these methods, we apply similar ideas to policy gradient methods.

Furthermore, [4] investigated the application of mirror descent to TD(λ). Much of their justification and observed results seem like they could carry over to policy gradient methods, which was the basis of our decision to formulate Mirrored PG.

In addition, policy gradient methods do not usually deal with constraints. However, [8] gives great reason for adding the ability to deal with constraints–specifically in the context of safety. While [8] focuses on natural actor critic, we use their ideas in the Mirrored PG formulation for policy gradient methods in general.

2 Background

2.1 Mirror Descent

Mirror map Let $\Phi : \mathbb{R}^n \to \mathbb{R}$. Φ is a *mirror map* if it satisfies the following:

(i) Φ is strictly convex and differentiable

(ii) The gradient of Φ is surjective, i.e. $\nabla\Phi(\mathbb{R}^n) = \mathbb{R}^n$.

(iii) The gradient of Φ diverges:

$$\lim_{x\to+\infty} \|\nabla\Phi(x)\| = +\infty$$

Bregman divergence The *Bregman divergence* associated with Φ is:

$$D_\Phi(x, y) \triangleq \Phi(x) - \Phi(y) - \langle\nabla\Phi(y), x - y\rangle$$

Note that many metrics are special cases of Bregman divergence, for example Euclidean distance when $\Phi(x) = \frac{1}{2}\|x\|_2^2$, or KL divergence when Φ is the entropy function.

Fenchel transform The Fenchel transform of $h : \mathbb{R}^n \to \mathbb{R}$ is denoted by h^*, and given by [6]:

$$h^*(y) \triangleq \sup_x\{\langle x, y\rangle - h(x)\}$$

A key property is that $\nabla h^* = (\nabla h)^{-1}$, or in other words, $x = \nabla h^*(\nabla h(x))$.

The (unconstrained) *mirror descent* strategy to minimize f is:

$$\theta_{t+1} := \nabla\Phi^*(\nabla\Phi(\theta_t) - \alpha\nabla f(\theta_t))$$

The original idea and explanation of mirror descent in [5] came from ideas in functional analysis, where the gradient of the mirror map (or more generally the Fréchet derivative) maps between primal and dual spaces. However, more recent work in [1] gives a simpler and more intuitive explanation.

The key observation is that the gradient descent update can be written in terms of the proximal operator:

$$\theta_{t+1} := \arg\min_u\{\langle u, \nabla f(\theta_t)\rangle + \frac{1}{2\alpha}\|u - \theta_t\|_2^2\}$$

With mirror descent, the update is the same, but we replace the Euclidean metric with the more general Bregman divergence (based on the mirror map):

$$\theta_{t+1} := \arg\min_u\{\langle u, \nabla f(\theta_t)\rangle + \frac{1}{\alpha}D_\Phi(u, \theta_t)\}$$

3 Sparse PG

In order to encourage sparse parameters, we can add an ℓ_1 regularization term to the policy gradient objective as follows:

$$\underset{\theta}{\text{maximize}} \quad \eta(\theta) - \lambda\|\theta\|_1$$

where $\eta(\theta)$ is the main policy gradient objective [7]. Note that this objective cannot be handled by stochastic gradient descent, as the objective is not differentiable due to the ℓ_1 term. There are many ways to resolve this issue, but we shall use the proximal gradient method due to simplicity.

Let $\widetilde{\nabla}\eta(\theta)$ denote a stochastic gradient for the policy gradient objective (e.g. $\gamma^t G_t \nabla_\theta \log \pi(A_t|S_t, \theta_t)$ for REINFORCE [9]). The SGD update now becomes:

$$\theta_{t+1} := \text{prox}_{\alpha\lambda\|\cdot\|_1}\big(\theta_t + \alpha\widetilde{\nabla}\eta(\theta)\big) \tag{1}$$

Elementary computations show that:

$$\text{prox}_{\alpha\lambda\|\cdot\|_1}(x) = (x - \alpha\lambda)_+ \circ \text{sign}(x) \tag{2}$$

where $\circ$ refers to the Hadamard product and $(\cdot)_+$ replaces all negative components with 0. Thus, combining (1) and (2) gives us:

$$\theta_{t+1} := (\theta_t + \alpha(\widetilde{\nabla}\eta(\theta) - \lambda))_+ \circ \text{sign}(\theta_t + \alpha\widetilde{\nabla}\eta(\theta)) \tag{3}$$

Replacing the update step of a policy gradient method with (3) will give a sparse version of the method.

4 Mirrored PG

Using mirror descent with a mirror map defined by a p-norm has been shown to work especially well in high-dimensional and noisy settings. Specifically, [2] shows this with linear regression and binary classification. Additionally, [4] exhibited success with mirror descent in TD(λ) using a multitude of mirror maps. As a result, we apply mirror descent to policy gradient methods in hopes of achieving similar success.

Traditionally, policy gradient methods do not have constraints. However, constraints can be very useful so we will include the ability to handle them. Let $\mathcal{X}$ be a convex set such that we require $\theta \in \mathcal{X}$. A couple practical examples of $\mathcal{X}$ include the convex hull of some values (useful if we want to constrain θ for safety purposes) and the unit simplex (useful when dealing with mixtures). Alternatively, we can set $\mathcal{X} = \mathbb{R}^n$ if no constraints are desired.

Let Φ denote a mirror map, and Φ^* its Fenchel transform. We will denote the projection operator on our constraint set by $\Pi_{\mathcal{X}}^{\Phi}(y) := \arg\min_{x\in\mathcal{X}} D_\Phi(x, y)$.

We can now modify the policy gradient update as follows to create Mirrored PG:

$$\begin{aligned} w_{t+1} &:= \nabla\Phi^*\big(\nabla\Phi(\theta_t) + \alpha\widetilde{\nabla}\eta(\theta_t)\big) \\ \theta_{t+1} &\in \Pi_{\mathcal{X}}^{\Phi}(w_{t+1}) \end{aligned}$$

Replacing the update step of a policy gradient method with the above will give a mirrored version of the method.

4.1 p-norm Mirror Map

Using Mirrored PG requires one to choose a mirror map. As we have previously mentioned, a p-norm is one such choice that has shown its mettle in a variety of situations.

Let $p, q \in \mathbb{R}$ s.t. $\frac{1}{p} + \frac{1}{q} = 1$, and let $\Phi(x) := \frac{1}{2}\|x\|_p^2$. It is immediate to verify that Φ is a valid mirror map. As shown in [4], the Fenchel transform and gradients are as follows:

$$\Phi^*(x) = \frac{1}{2}\|x\|_q^2$$

$$(\nabla\Phi(x))_j = \frac{\text{sign}(x_j)|x_j|^{q-1}}{\|x\|_q^{q-2}}$$

$$(\nabla\Phi^*(x))_j = \frac{\text{sign}(x_j)|x_j|^{p-1}}{\|x\|_p^{p-2}}$$

5 Empirical Results

5.1 Cart Pole Domain

To test Sparse PG and Mirrored PG, we chose to use a noisy version of the Cart Pole domain, which we describe here.

Cart Pole[1] is a classic domain used in the RL literature. In this domain, a pole is attached (by an un-actuated joint) to a cart which moves along a friction-less track. The state space is continuous, and consists of the cart's x position, the velocity in the x direction, the angle of the pole with respect to the vertical axis, and the speed with which the pole's angle is changing. The actions are applying a $+1$ or -1 force to the cart (i.e. pushing it right and left). A $+1$ reward is provided every timestep, and the episode ends when the pole drops too low or the cart goes out of bounds.

Additionally, two features with uniformly random values are provided as part of the state. This is done to simulate noise and irrelevancy, which is present in many complex domains. For example, if one is using pixel values as features in learning an Atari game, many pixels would be noisy and irrelevant.

We used REINFORCE as the base policy gradient method with linear function approximation, a softmax policy, a learning rate of $.0003$, and 1 as the discount factor.

5.2 Sparse PG

The Sparse PG formulation was tested on noisy Cart Pole, with the results shown in Figure 1(a). Using ℓ_1 regularization shows significant improvement over the baseline REINFORCE.

5.3 Mirrored PG

Mirrored PG with a p-norm mirror map was also tested on noisy Cart Pole (Figure 1(b)). Greater values of p lead to increased performance, with performance starting to drop at $p = 30$.

Furthermore, Figure 1(c) shows a comparison of all the methods (baseline REINFORCE, sparse REINFORCE, and mirrored REINFORCE), where Mirrored PG beats out the other methods quite significantly.

5.4 Removing Noise

Additionally, the methods were run on normal Cart Pole (i.e. with the two noisy features removed), as shown in Figure 1(d). As expected, Sparse PG performs terribly due to there being no irrelevant features; after all, we do not want any sparsity in this case. Performance picks back up as the regularization parameter is decreased. On the other hand, Mirrored PG maintains its lead and continues to outperform the other methods.

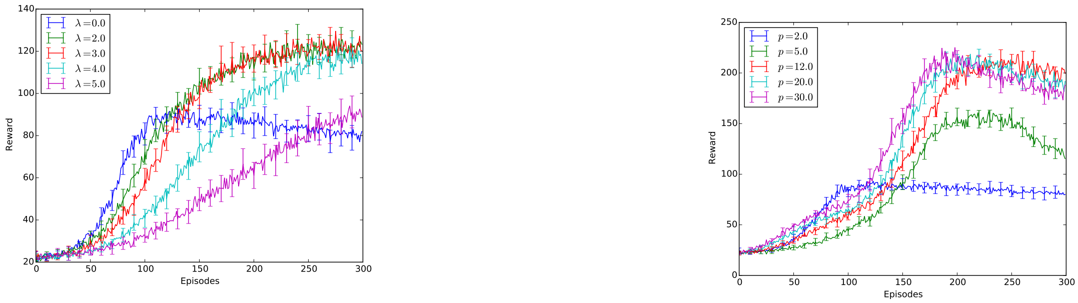
This shows a weakness of Sparse PG, in that it is very sensitive to its regularization parameter which requires careful tuning. Making the regularization parameter too large can completely halt learning.

6 Conclusion

In this paper, we introduced the Sparse PG and Mirrored PG extensions to policy gradient methods. Furthermore, we demonstrated their effectiveness in the Cart Pole environment, with the Mirrored PG p-norm formulation showing itself to perform the best. Indeed, with the great many possibilities of mirror maps, Mirrored PG forms a very flexible and

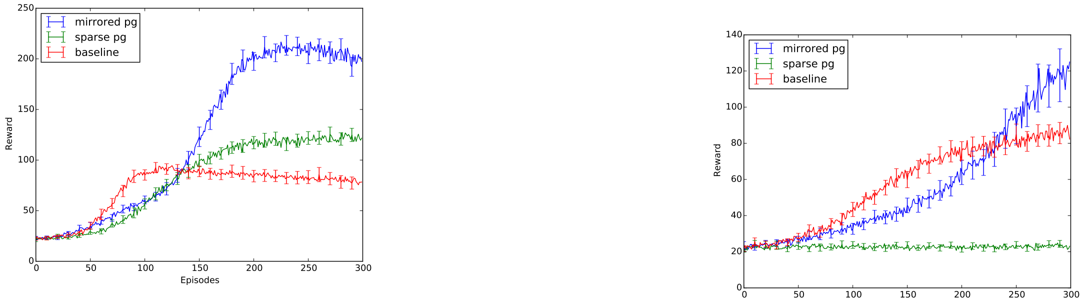
---

[1] The OpenAI Gym implementation of Cart Pole was used in these experiments.



(a) Sparse PG with different regularization parameters (λ). Note that $\lambda = 0$ corresponds to the vanilla REINFORCE baseline.

(b) Mirrored PG with different p values. Note that $p = 2$ corresponds to the vanilla REINFORCE baseline.



(c) Comparison of vanilla REINFORCE with the sparse and mirrored formulations of it. For Sparse PG, $\lambda = 3.0$, and Mirrored PG uses $p = 12$ (i.e. the best parameter values from the above graphs).

(d) All the methods (as in (c)) compared on normal Cart Pole.

Figure 1: Results on the Cart Pole domain. Error bars are 95% confidence intervals.

general extension to policy gradient methods. We leave testing more complex domains and evaluating other mirror maps to future work.

References

[1] A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

[2] C. Gentile. The robustness of the p-norm algorithms. *Machine Learning*, 53(3):265–299, 2003.

[3] J. Z. Kolter and A. Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 521–528. ACM, 2009.

[4] S. Mahadevan and B. Liu. Sparse q-learning with mirror descent. *arXiv preprint arXiv:1210.4893*, 2012.

[5] A. Nemirovsky and D. Yudin. Problem complexity and method efficiency in optimization. 1983.

[6] F. Nielsen. Legendre transformation and information geometry. Technical Report CIG-MEMO2, September 2010. http://www.informationgeometry.org.

[7] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

[8] P. S. Thomas, W. C. Dabney, S. Giguere, and S. Mahadevan. Projected natural actor-critic. In *Advances in Neural Information Processing Systems*, pages 2337–2345, 2013.

[9] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Hierarchical State Abstraction Synthesis for Discrete Models of Continuous Domains

Jacob Menashe and Peter Stone
{jmenashe,pstone}@cs.utexas.edu
The University of Texas at Austin
Austin, TX USA

Abstract

Reinforcement Learning (RL) is a paradigm for enabling autonomous learning wherein rewards are used to influence an agent's action choices in various states. As the number of states and actions available to an agent increases, so it becomes increasingly difficult for the agent to quickly learn the optimal action for any given state. One approach to mitigating the detrimental effects of large state spaces is to represent collections of states together as encompassing "abstract states".

State abstraction itself leads to a host of new challenges for an agent. One such challenge is that of automatically identifying new abstractions that balance generality and specificity; the agent must identify both the key similarities and differences between states that are relevant to the agent's goals, while ignoring unnecessary details from the environment. We call this problem of identifying abstract states the Abstraction Synthesis Problem.

In this work we propose the Recursive Cluster-based Abstraction Synthesis Technique (RCAST), a new method for abstraction synthesis. We provide the algorithmic details of RCAST and its subroutines, and compare the general properties of RCAST with those of alternative abstraction synthesis algorithms. Finally we show that RCAST enables RL agents to quickly and accurately identify helpful transactions in a variety of RL domains with minimal need for expert configuration.

Keywords: Reinforcement Learning, Model-based Reinforcement Learning, Hierarchical Reinforcement Learning, Knowledge Representation, Active Learning

1 Introduction

The efficiency with which an AI agent learns the dynamics of a domain is heavily influenced by that agent's internal representation of the world. This is especially true for complex, multi-faceted domains that are often found in the real world. For this reason, AI researchers are often forced to wrestle with the so-called *curse of dimensionality* wherein the complexity of a domain scales exponentially with the number of variables used to describe it.

A traditional solution to this problem is through the use of hierarchical layers of abstraction; rather than painstakingly learning about every individual state possible in some domain, an agent can instead group large numbers of states together and consider only this abstract representation during the learning process. While such representations are merely helpful for simpler domains, for domains over real-valued variables (such as one's position in continuous space) abstractions are all but necessary to apply any form of machine learning.

In this work we present an abstraction synthesis technique that leverages hierarchical discretization and recursive clustering techniques to efficiently build abstractions that balance generality and specificity, allowing an AI agent to quickly build useful and computationally simple models of the environment. We refer to this approach as *the Recursive Cluster-based Abstraction Synthesis Technique* (RCAST). In Section 2 we describe the existing work related to RCAST as well as the general context of its use in learning problems and the learning frameworks upon which it depends. In Section 3 we describe RCAST in detail and in Section 4 we conclude and discuss future work.

2 Background and Related Work

In this paper we focus our discussion on state abstraction synthesis in the context of Reinforcement Learning (RL). Thus we begin our discussion of background literature with a brief overfiew of RL, and then proceed to cover the related work on state abstraction.

2.1 Markov Decision Processes

The Markov Decision Process (MDP) is a mathematical description of a task that is central to RL theory. An MDP M is defined by a 5-tuple $\langle S, A, P, R, \gamma \rangle$ with state space S, action space A, transition distribution function P, reward function R, and discount factor γ. When $S = S_1 \times S_2 \times \ldots \times S_n$ we say that S and M are *factored*, and that M is a Factored MDP (FMDP). Reinforcement Learning methods seek to learn optimal policies through exploring and exploiting the state-action space of an MDP.

2.2 Model-based Reinforcement Learning

Model-based reinforcement learning is a branch of reinforcement learning in which an agent uses a model to predict the effects of its actions in the environment. In effect, while classical Reinforcement Learning is concerned with learning a value function dependent upon R, model-based reinforcement learning is additionally concerned with invoking (and possibly learning) an approximation of P.

In our work we consider models based on Dynamic Bayesian Netowrks (DBNs). Earlier work by Dean and Kanazawa [1] describes how an FMDP can be modeled compactly as a set of DBNs. In this work RCAST encodes the effects of each action with a single DBN, where the DBN is a two-layer directed bipartite graph with nodes corresponding to state variables and edges denoting dependence under the relevant action.

Jonsson and Barto [5] show how a DBN can be encoded with a set of Conditional Probability Trees (CPTs), with each CPT encoding the dependencies of a single state-variable/action pair. The tree's internal nodes are each tagged with a single state variable which indicates dependency with respect to the CPT's action. A CPT can be *refined* by adding internal nodes over time as these dependencies are discovered in the environment. Each such node is thus a *refinement*.

2.3 State Abstraction Synthesis

Small, finite, and factored state spaces give rise to useful and intuitively defined state abstraction mechanisms. Jong and Stone [3] propose a method for state abstraction in such factored state spaces through identification of so-called "irrelevant" factors. For instance, if the state space S has factors X and Y, then an abstract state might be a particular assignment $X = x_0$ with no assignment for Y. In this case, the abstract state space S' consists of $|X|$ abstract states each encompassing $|Y|$ primitive states. Jonsson and Barto [4] use decision tree models of the state space toward a similar end, where each branch encodes an assignment of variables to values, and omitted variables represent those that are irrelevant for a particular action model.

Rather than defining abstractions in terms of critical values, there has been ample work on defining abstractions in terms of their relevance to "macro" actions using the options framework [15]. "Bottleneck" options are one such example where the state space is divided into regions on either side of highly-traversed intermediate states ("bottleneck" states). The initiation and termination sets of such options each designate two distinct abstract states that can be used for planning in lieu of the primitive state space [11, 10, 14].

State abstractions can be more difficult to synthesize in domains with continuous-valued state variables. Due to the small likelihood of visiting a single continuous state value assignment multiple times, an agent must instead attempt to visit neighborhoods about such values for effective planning. Planning with neighborhoods raises the challenge of determining the appropriate size and shape of such neighborhoods. Option-based state abstraction extends naturally into continuous domains, however this does not relieve

```
1: function RCAST($F_i, F_o, D, Q, L$)
2:     $D' \leftarrow D \sqcap Q$
3:     $C' \leftarrow$ X-Means($D'$)
4:     $C' \leftarrow F_i(C)$
5:     $T \leftarrow$ H-Part($Q, C'$)
6:     $L \leftarrow$ SplitLeaf($L, F_i, T$)
7:     return $L$
8: end function
```

(a) A concrete implementation of RCAST.

```
function H-PART($Q$, $C$)
    $T \leftarrow$ an empty KD-Tree with label “undecided”
    $C' \leftarrow \{c \sqcap Q | c \in C, c \sqcap Q \neq \emptyset\}$
    if One cluster $c$ in $C'$ then
        $T.label \leftarrow c.label$
    else if Diameter($Q$) $\leq 1$ then
        $T.label \leftarrow$“multi”
    else
        Recursively split and label $T$ with H-Part by a factor of 10
    end if
    return $T$
end function
```

(b) H-Part: A hierarchical cluster-based spatial partitioning algorithm.

Algorithm 1: RCAST and its primary subroutine H-Part

the aforementioned difficulty in identifying continuous neighborhoods. Such neighborhoods can be classified using a traditional supervised learning approach [8], but this relies on large numbers of sample trajectories and predefined classes used to label the samples. Ultimately, any supervised learning approach to abstraction synthesis will require repeated intervention from an expert to provide training labels.

Alternative approaches rely on iteratively dividing the state space into half-spaces using orthogonal hyperplanes [6, 13, 9, 7, 2]. Such iterative half-space (IHS) approaches identify optimal hyperplanes for splitting some space one at a time until all of the splits necessary to fully describe the space's dynamics have been identified. While this technique can achieve arbitrary levels of precision, it invariably results in creating unnecessary abstract states as a side-effect of the iterative halving process. Moreover, when multiple half-spaces are required for meaningful separation of datapoints, the initial splits must be performed with limited statistical indication of their relevance. Thus the algorithm must split aggressively in anticipation of high quality abstractions many iterations in the future, and at the same time split conservatively to avoid creating abstractions that are harmful to the learning process. The overall effect is that such algorithms tend to be either sample-inefficient or inaccurate.

Fayyad and Irani [2] tackle the state abstraction problem with unidimensional continuous factors by hierarchically splitting continuous intervals into two parts at a time, but even this approach suffers from creating unnecessary abstract states and is poorly suited to continuous factors over multiple dimensions.

Our work improves upon that of Fayyad and Irani by both removing the need for creating unnecessary abstract states and enabling abstraction over continuous factors of arbitrary dimensionality. Section 3 provides a more detailed description of these differences.

3 The RCAST Algorithm

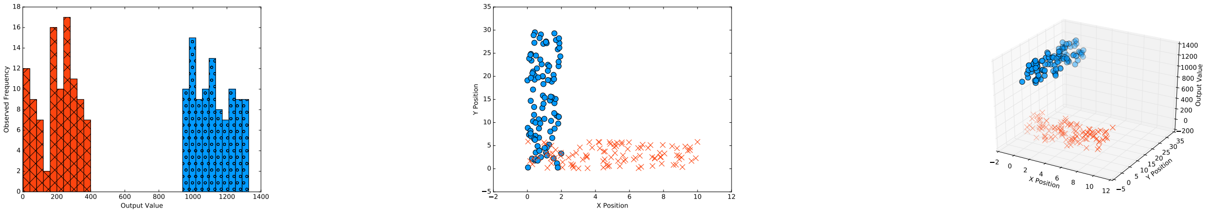
This section introduces RCAST, the main contribution of the paper. We will describe a concrete implementation of RCAST in Section 3.1 using the model framework and action selection mechanisms described by Menashe and Stone [12] as the T-UCT algorithm. RCAST will thus augment T-UCT with the ability to model changes in state in the context of continuous factors.

At each timestep t, T-UCT collects a datapoint consisting of the state s_{t-1}, the last action a which was taken, and finally the consequent state s_t. T-UCT then identifies a collection of CPTs in its model which correspond to a and adds this latest datapoint $d_t = \langle s_{t-1}, a, s_t \rangle$, analytically determines each unique leaf that matches s_{t-1} from each CPT. For each such leaf l, T-UCT adds d_t to its dataset, and then analyzes that dataset to determine if it can be better described by splitting the CPT on some factor at l. Jonsson and Barto [5] describe this process in detail.

T-UCT only allows for such splits to occur on tabular variables. With this restriction, T-UCT splits on a factor f at leaf l by creating one child of l for each possible value of f. Clearly, such a splitting scheme cannot be applied when the domain of f is continuous due to its infinite cardinality. In this work we will apply RCAST to the task of filling in this gap, allowing efficiently-computed and informative splits over continuous factors.

3.1 Algorithm Description

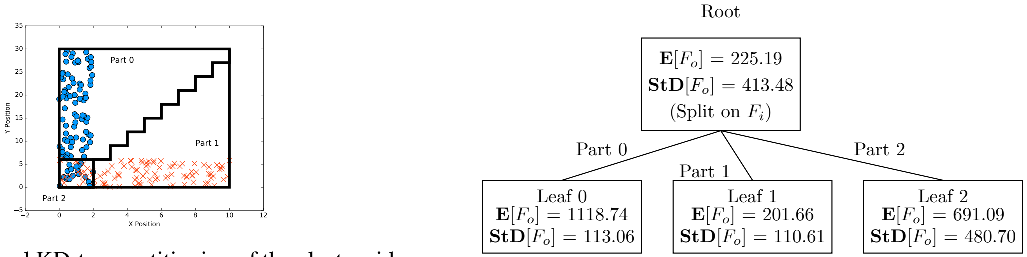
RCAST's purpose is to analyze observed dynamics in a given environment and identify key areas of the environment that exhibit similar dynamics. RCAST's analysis is then leveraged through model refinements which allow an agent to use its experiences to knowledgeably plan its actions. Algorithm 1a provides pseudo-code for the main RCAST implementation. Line 1 defines the basic inputs to the algorithm including an *input factor* F_i, an *output factor* F_o, a dataset D, and an orthotope Q describing the bounds of F_i. In calling this function we assume that changes in F_o depend on F_i when F_i's value falls within Q. We will be analyzing D to identify the specifics of this relationship. The final input variable, L, is the CPT leaf which will be refined based on this analysis.



(a) A histogram of values taken from some hypothetical dataset. Values are derived from the next-state of each datapoint for some hypothetical factor F_o. The two patterns represent the two optimal clusters.

(b) A plot of the same dataset using previous-states and factor F_i where F_i happens to be 2D. The two colors and shapes represent the same optimal clusters.

(c) A 3D plot of the previous two dataset projections, where each point is the concatenation of the the F_i and F_o values shown above. The XY plane represents F_i, and the Z axis represents F_o.


Part 0
Part 1
Part 2
Y Position
X Position
Root
$\mathbf{E}[F_o] = 225.19$
$\mathbf{StD}[F_o] = 413.48$
(Split on F_i)
Part 0
Part 1
Part 2
Leaf 0
$\mathbf{E}[F_o] = 1118.74$
$\mathbf{StD}[F_o] = 113.06$
Leaf 1
$\mathbf{E}[F_o] = 201.66$
$\mathbf{StD}[F_o] = 110.61$
Leaf 2
$\mathbf{E}[F_o] = 691.09$
$\mathbf{StD}[F_o] = 480.70$

(d) A hypothetical KD-tree partitioning of the clusters identified in Figure 3.1c. Lines on the plot designate the separate partitions. Partition indices correspond with leaf indices in Figure 3.1e.

(e) The induced CPT refinement based on the partitions in Figure 3.1d. Statistical values are computed from the observed data.

Figure 3.1: An example of the RCAST algorithm being applied to some hypothetical dataset.

In Line 2 we see that the use of the $\sqcap$ operator applied to D and Q. This operator restricts the dataset to those datapoints whose predecessor states' value assignments for F_i fall within the bounds of Q. Intuitively, this means that each state-action-state sequence "started" in Q. Line 3 then clusters this subset of points using X-Means Clustering, which produces as many clusters as necessary to maximize the BIC score of the clusters. Line 4 projects the clusters' datapoints into the value space Q of F_i so that Q can be partitioned in accordance with these clusters. Line 5 then hierarchically partitions Q according to C' using Algorithm 1b as the partitioning subroutine. Algorithm 1b produces a KD-Tree with leaves labeled according to the clusters they encompass. The trees are recursively discretized by a factor of 10.

In the context of T-UCT, the partitions produced by Algorithm 1a are used to split leaves of T-UCT's CPT model. Jonsson and Barto [4] show that a CPT leaf can be split along some input factor F_i' by creating one child leaf per value of that factor. In a sense, the discrete value space of F_i' is partitioned into singleton subspaces. In the context of a continuous and multidimensional F_i, Algorithm 1b produces a labeled tree where the set of leaves for a particular label defines a subspace of F_i. Thus, we can refine a CPT model over a continuous, multidimensional F_i by partitioning its value space and then creating one new CPT leaf per label. In Line 6 the algoriths splits L in just this manner, which refines the overarching transition model. Figure 3.1 provides a hypothetical example of this process.

4 Conclusion and Future Work

In this work we have described RCAST, a new method for synthesizing abstract states based on observed data. RCAST requires little expert knowledge which renders it preferable to comparable supervised learning techniques in the context of efficient and accurate abstraction synthesis. Moreover, RCAST allows refinement of discrete models with less data than IHS methods.

A primary focus for future work lies in modifying the clustering subroutine which RCAST depends upon. While X-Means Clustering achieves the basic requirements of RCAST, there exists a vast array of clustering algorithms in the present literature which may be better suited to the problem of abstraction synthesis. For instance, X-Means is reliant upon centroid distance comparisons and is thus biased toward ellipsoid clusters; it may instead be advantageous to employ a hierarchical clustering algorithm that is better equipped to generate irregularly shaped clusters.

Additionally, we note that the discretization mechanism of Algorithm 1b arbitrarily divides bins at equal intervals along the spatial boundaries, similar to a naive tile coding. It may be possible to improve the accuracy of Algorithm 1b by allowing for multiple overlapping tilings.

Finally, we note that while RCAST has been shown to improve model induction, its contributions to general HRL and the pursuit of discounted reward should be studied further. Thus, a promising area of future work lies in the incorporation of RCAST in general HRL tasks using more traditional RL metrics such as the accumulation of episodic and lifetime reward.

References

[1] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational intelligence*, 5(2): 142–150, 1989.

[2] Usama Fayyad and Keki Irani. Multi-interval discretization of continuous-valued attributes for classification learning. 1993.

[3] Nicholas K Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *IJCAI*, volume 8, pages 752–757, 2005.

[4] Anders Jonsson and Andrew Barto. A causal approach to hierarchical decomposition of factored MDPs. In *Proceedings of the 22nd international conference on Machine learning*, pages 401–408. ACM, 2005.

[5] Anders Jonsson and Andrew Barto. Active learning of dynamic Bayesian networks in Markov decision processes. In *Abstraction, Reformulation, and Approximation*, pages 273–284. Springer, 2007.

[6] Anders Jonsson and Andrew G Barto. Automated state abstraction for options using the u-tree algorithm. *Advances in neural information processing systems*, pages 1054–1060, 2001.

[7] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, volume 96, pages 202–207. Citeseer, 1996.

[8] George Konidaris and Andrew G Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in neural information processing systems*, pages 1015–1023, 2009.

[9] Bing Liu, Yiyuan Xia, and Philip S Yu. Clustering through decision tree construction. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 20–29. ACM, 2000.

[10] Amy McGovern and Andrew G Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. 2001.

[11] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cutdynamic discovery of sub-goals in reinforcement learning. In *European Conference on Machine Learning*, pages 295–306. Springer, 2002.

[12] Jacob Menashe and Peter Stone. Monte Carlo Hierarchical Model Learning. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 771–779. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[13] John R Quinlan et al. Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, volume 92, pages 343–348. Singapore, 1992.

[14] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 212–223. Springer, 2002.

[15] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

Learn to Survive by Deep Reinforcement Learning

Naoto Yoshida
GROOVE X
Nihonbashi-Horidomecho, Chuo-ku, Tokyo, Japan
naoto.yoshida@groove-x.com

Abstract

The dynamic stability of the internal environment is one of the main characteristics of animals. In our work, we revisit the work of the Ashby's homeostat, that is an early computational model of the biological homeostasis with substantial simplification. Our recent study suggests that recapturing the homeostat from the view point of the probabilistic framework connects the maximization of the survival probability by using the variational lower bound with the computational reinforcement learning (RL) theory with given form of reward functions. In this paper, we extend our previous work and solved the classical and *visual* two-resource problem by using deep neural networks. Even though our agents only have primitive actions, we observe that the switching behavior between two food resources.

Keywords: Reinforcement Learning, Reward, Homeostat, Foraging, Survival

1 Background

Survival strategies are essential for biological agents in the wild [17, 2]. Many researchers have developed various types of survival agents since the early days of artificial intelligence [1, 14, 13, 6, 3, 5, 4]. Reward stimuli were introduced in most of the preceding research studies, with a reward function being necessary for the computational reinforcement learning (RL) paradigm. However, almost all of the previous studies on survival adopted hand-crafted reward functions that do not guarantee the intended behaviors, which are the survival strategies in this case. The reward and objective function given by the designer may work well in simple RL tasks. However, for more complex tasks and life-long learning settings, a badly hand-crafted reward function may have damaging effects on system performance in the end.

In our recent study [16, 15], we formally re-formulated the survival problem from the view point of probabilistic framework. Similar ideas from the view point of optimal control theory are suggested in ethology [11, 7, 6], but our formulation include more general case that the environment may not be Markovian. In our previous study, we empirically examined our reward function in the toy-domain with table representations [16]. However, the capability of our formulation in more general environment with function approximation remained unknown. In this work, we examine more complex environment constructed by following the classical two-resource problem.

2 Problem Formulation and Homeostatic Reward Functions

In the classical model [6], an animal has several variables which is observed by the animal and have some importance for sustaining its life (for example, the water level and some nutrient level). These variables are called *physiological state* s. And if the physiological state exceed the predefined region (*physiological space*), the agent immediately dies. Agents also has a *perceptual state* which is the perception of the environmental stimuli (vision, touch, etc.). The combined physiological state and perceptual state is called the *motivational state* [6], that corresponds to the observation in the context of the partially observable reinforcement learning (PORL).

The threshold between death and live is hard and it prevented the probabilistic treatment of this problem and online learning [1, 11, 6]. In our recent study, we extended this hard threshold by softening the death/live boundary of the physiological space [16, 15]. In our framework, we introduce a binary survival flag $A \in \{0, 1\}$. $A = 1$ represents that the agent is 'live' and otherwise 'dead'. And then, we introduce the temporal survival probability at the time t as $P(A_{t+1} = 1|s_t)$. This probability represent the probability that the agent is alive at the next time step $t + 1$. The T-step survival probability is immediately represented as

$$P(A_1 = 1, A_2 = 1, \ldots A_T = 1|\pi, A_0 = 1), \tag{1}$$

with agent's behavioral policy π.

In order to solve above problem in the reinforcement learning (RL) setting, we introduced the homeostatic reward function

$$r_t = \log P(A_{t+1} = 1|s_t). \tag{2}$$

The detailed derivation of this reward is explained in the another place [16, 15]. Intuitively, the maximization of the cumulative sum of this reward function (and some applicable transformation suggested by Ng et al. [10]) corresponds to the maximization of the original objective (equation 1) from the variational lower bound. Even though we have to introduce the temporal survival probability, the reward signal becomes fully intrinsic signals and this probability function is more intuitive than that of the classical reward function because this function have an explicit meaning. We have to note that this reward function is the *sufficient* reward function for survival, but not the *optimal* reward function. However, this reward can provide the baseline for future extensions by the shaping reward [10] and so on.

3 Experiment

The two-resource domain or Spier's domain is a classical domain for the homeostatic agent [12, 5]. The agent is required to gather two kind of nutrient resources and balance the level of the both nutrients ($s_{\mathrm{red}}, s_{\mathrm{blue}}$). Importantly, agents initially do not know what perceptual stimuli are related to the nutrient resource in both of our two experiments. Therefore the agent has to associate these objects (perceptual state) with changes in resource levels (physiological state). Also, in our original reward setting, agents never receive positive rewards when they take nutrient resources and the amount of rewards for nutrient-capture depend on the agent's resource level. Therefore, the agent initially doesn't have any knowledge of nutrient resource internally and then this experiment is fundamentally different from task-oriented problems like the "food capturing".

3.1 Two-Resource Domain with Low Dimensional Input

Figure 1(a) represents our implementation of the two-resource domain. The size of the field is 480x480 pixels. In the field, there are two kind of food objects (red and blue) and the number of each food is 15. Agents (a turtle in the figure) have three range finders for red food, blue food and walls (the edge of square field) to detect distance from objects. Because range finders have detection limits, agents only can observe the partial information about the field. If the distance between the center of the agent and the (red or blue) food object is less than 30 pixel (that is, if the agent ”touches” the food), the nutrient resource corresponding to the food color is supplied by 1.0 and that food disappears (food is eaten). When a food is eaten, new food object is generated at a random position to maintain the number of the food in the field. Nutrient resource levels are initialized with $s_{\mathrm{red}} = s_{\mathrm{blue}} = 0$ and decreased by 0.01 every step.

Similar experiment was treated by [5] as a RL experiment, but they defined high-level actions, those are ”approaching” and ”consuming” each visible target. In our experiment, the agent only has three primitive actions “Rotate Clockwise”, “Rotate anti-Clockwise” and “Forward”. Rotation actions rotate an agent by 15 degree at the current position, an agent proceeds 30 pixels when “Forward” action is selected.

In this experiment, we set the temporal survival probability as the function of the two nutrient resource level (red and blue)

$$P(A_{t+1} = 1 | s_{\mathrm{red}}, s_{\mathrm{blue}}) = e^{-k(s_{\mathrm{red}}^2 + s_{\mathrm{blue}}^2)} \tag{3}$$

and the reward is generated by taking the natural logarithm of this probability. In our experiment $k = 0.05$ was chosen in this experiment. If red or blue nutrient resource level of the agent exceed the range $[-10.0, 10.0]$, that episode is terminated. Then the next episode starts with $s_{\mathrm{red}} = s_{\mathrm{blue}} = 0$ and the agent, red food, blue food are randomly and uniformly settled in the field. Because this RL task is continual, the agent can survive arbitrarily long time once the successful survival policy is obtained. Therefore, we forced to the terminate episodes if the agent survived 10,000 time steps

In this experiment, we construct an RL agent by Q-learning with the function approximation. We used a three-layer perceptron as the function approximator. The neural network has input units, two thousand hidden units with the rectified linear (ReLU) activation function, and three linear output units. The number of the output unit correspond to the number of actions. The input of the network is a vector of concatenated sensor inputs, that are the range finder (wall, red, blue) and the nutrient resource levels (red and blue).

The result is shown in Figure 1. In the left figure (a), the vertical axis represents the survival time of the agent in the domain, the horizontal axis represents the number of episodes. This figure clearly represents that the agent initially cannot survive longer than 3000 steps, but finally learned to survive in the given environment. Figure 1(c) is an example of the red and blue resource levels in an agent during the 100-th episode. We can observe that the agent can balance the red and blue resources. Interestingly, we can observe that the agent automatically “switches” between two behavior mode *forage red* and *forage blue*. These result suggests that the behavior of our agent, that are foraging, resource balancing and behavioral switching were automatically emerged from the reward function with the internal signal of the agent. And importantly, these behaviors are naturally and theoretically integrated for the survival of the agent.

3.2 Visual Two-Resource Domain

We scaled the two-resource problem to treat the vision input. The 3D environment was constructed by LIS (Life-in-Silico) framework developed by [9]. LIS is a general-purpose reinforcement learning framework with Unity game engine. Using this framework, we can develop an arbitrary 3D (and 2D) RL environment with the dynamics engine using Unity editor tools.

The overall view of our environment is shown in Figure 2(a). These settings are similar to that of the previous experiment but now everything is in 3D. The approximately 12x12[m^2] field is surrounded by mountains, and the bottom of the field is covered by the grass texture. The square field is surrounded by four invisible walls (visualized as green frames in the figure) to prevent agent from being embedded in the mountains and seeing the outside of the field. As is in the previous experiment, there are two kind of food that have different nutrient property (red and blue food) and there are 15 units for each. These food are represented by small cubes and scattered uniformly and randomly over the entire field.

When the front area of the agent touches the food cube, agent’s nutrient resource corresponding to the food color is supplied by 0.3 and the touched food is erased. Then a new food object is generated at the random position to maintain the number of food in the field. The color (RGB) vision information is provided from the single front camera on the agent and then the vision information is scaled into 32x32x3 pixel images (an example is shown in Figure 2(b)). In total, the agent is provided the current and previous vision input, and the internal resource levels.

Nutrient resource levels are decreased by 0.01 every step. Similar to the previous experiment, the agent has three primitive actions ”Rotate Clockwise”, ”Rotate anti-Clockwise” and ”Forward”. The reward is given as the function of the two nutrient resource levels (red and blue) that is the same with the previous experiment, $k = 0.01$ was chosen in this experiment. When the nutrient resource level of the agent exceeds the range $[-10.0, 10.0]$, that episode is terminated and the next episode starts with $s_{\mathrm{red}} = s_{\mathrm{blue}} = 0$. The agent, red food, blue food are randomly and uniformly settled over the entire field. Again, this is a continual task and the agent can survive arbitrarily long time when the successful survival policy is obtained. Therefore, we forced to terminate episodes when the agent survived 20,000 time steps in an episode.

We applied Deep Q-network (DQN) developed by Mnih et al. [8] in this task. DQN is composed of a convolutional neural network to efficiently treat the visual information. In the network, the visual information was encoded by two convolution layers and mixed with resource level signals. Resource levels are encoded by one fully connected layer. And finally the mixed layer was connected to the output layer. We adopted the epsilon-greedy exploration strategy, and decreased the exploration rate linearly along the time steps from 0.3 to 0.1.

The preliminary results were encouraging. Figure 3 show the results of this experiment. The vertical axis is the survival time steps of the agent at that episode, and the horizontal axis is the episode of the trial. The plot represents the average of 10 individual agents and we observed that all agents successfully learned to survive in the given environment (individual plots are not shown).

4 Conclusions

In this study, we extended our previous toy-scale experiment on the survival problem. We treated the low-dimensional and the high-dimensional two-resource problem and the results were encouraging. Environments in this paper were easy enough that the reactive agent can learn the survival policy. Our future step should be the treatment of more complex environments that requires the short-term memory to survive.

References

[1] William Ross Ashby. *Design for a Brain*. Springer Science & Business Media, 1960.

[2] R Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, UK, 1976.

[3] Stefan Elfwing, Eiji Uchibe, Kenji Doya, and Henrik I Christensen. Biologically inspired embodied evolution of survival. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2210–2216. IEEE, 2005.

[4] Mehdi Keramati and Boris S Gutkin. A reinforcement learning theory for homeostatic regulation. In *Advances in Neural Information Processing Systems*, pages 82–90, 2011.

[5] George Konidaris and Andrew Barto. An adaptive robot motivational system. In *From Animals to Animats 9*, pages 346–356. Springer, 2006.

[6] David McFarland and Tom Bösser. *Intelligent behavior in animals and robots*. MIT Press, 1993.

[7] David McFarland and Alasdair Houston. *Quantitative ethology*. Pitman Advanced Pub. Program, 1981.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[9] Masayoshi Nakamura and Hiroshi Yamakawa. A game-engine-based learning environment framework for artificial general intelligence. In *International Conference on Neural Information Processing*, pages 351–356. Springer, 2016.

[10] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[11] Richard Sibly and David McFarland. On the fitness of behavior sequences. *American Naturalist*, pages 601–617, 1976.

[12] Emmet Spier. *From reactive behaviour to adaptive behaviour: motivational models for behaviour in animals and robots.* PhD thesis, University of Oxford, 1997.

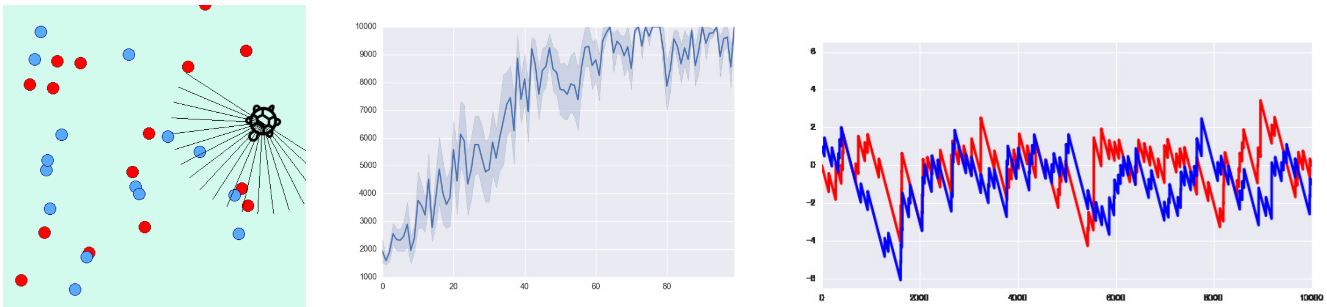
[13] Masanao Toda. The design of a fungus-eater: A model of human behavior in an unsophisticated environment. *Behavioral Science*, 7(2):164–183, 1962.

[14] W Walter. *The living brain.* Norton, 1953.

[15] Naoto Yoshida. Homeostatic agent for general environment. *in preparation*.

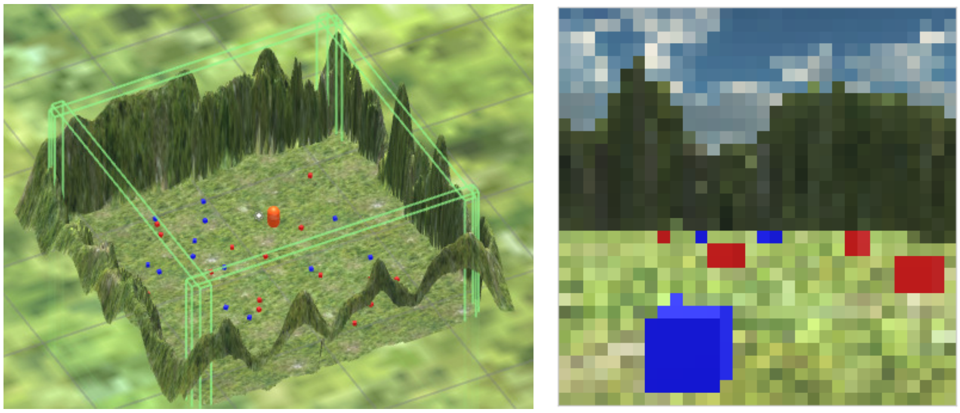
[16] Naoto Yoshida. On reward function for survival. *arXiv preprint arXiv:1606.05767*, 2016.

[17] John Zachary Young. *The Memory System of the Brain*. Oxford University Press, 1966.



(a) (b) (c)

Figure 1: (a) The two-resource environment with low-dimensional observations. There are agent and two nutrient resources (red and blue). The agent has three range finders for walls (the edge of the environment) and nutrient resources. The agent has two internal state that are red and blue nutrient resources levels. In order to survive, the agent needs to balance the level of both nutrient resources around zero (minimum: -10, maximum: +10). (b) The growth of survival time along the episode. The plot shows the mean and the standard deviation over ten individual run of the experiment. (c) The trajectory of two nutrient resource levels during the 100-th episode.



(a) Overview of the environment (b) Example of the vision input

Figure 2: Overview of the Visual Two-Resource Problem Environment

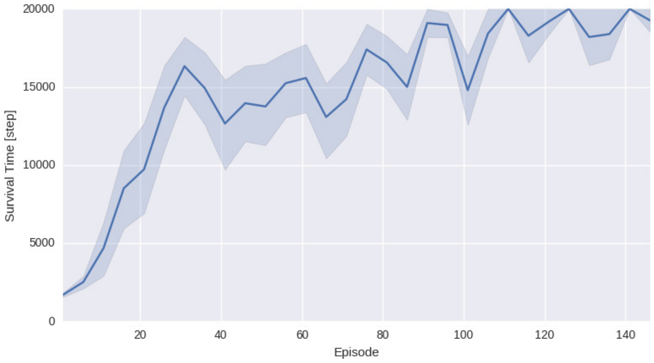


Figure 3: The survival time steps along the episodes. The plot is the average of the 10 individual agents. In this experiment, the maximum survival time is bounded by 20,000 steps.

Towards Stability in Learning-based Control: A Bayesian Optimization-based Adaptive Controller

Mouhacine Benosman
Mitsubishi Electric Research Laboratories (MERL)
Cambridge, MA 02139, USA
m_benosman@ieee.org

Amir-massoud Farahmand
Mitsubishi Electric Research Laboratories (MERL)
Cambridge, MA 02139, USA
farahmand@merl.com

Abstract

We propose to merge together techniques from control theory and machine learning to design a stable learning-based controller for a class of nonlinear systems. We adopt a modular adaptive control design approach that has two components. The first is a model-based robust nonlinear state feedback, which guarantees stability during learning, by rendering the closed-loop system input-to-state stable (ISS). The input is considered to be the error in the estimation of the uncertain parameters of the dynamics, and the state is considered to be the closed-loop output tracking error. The second component is a data-driven Bayesian optimization method for estimating the uncertain parameters of the dynamics, and improving the overall performance of the closed-loop system. In particular, we suggest using Gaussian Process Upper Confidence Bound (GP-UCB) algorithm, which is a method for trading-off exploration-exploitation in continuous-armed bandits. GP-UCB searches the space of uncertain parameters and gradually finds the parameters that maximize the performance of the closed-loop system. These two systems together ensure that we have a stable learning-based control algorithm.[1]

Keywords: Adaptive Control, Modular Control, Stability, Machine Learning, Bayesian Optimization, GP-UCB

---

[1] This paper is a summary of Benosman and Farahmand [2016]; Benosman et al. [2016].

1 Introduction

There are much common problems between the fields of control engineering and reinforcement learning, as is mainly practiced in computer science. A central problem in both is to make a sequence of decisions in order to ensure that a dynamical system behaves in a desired manner. This high-level similarity notwithstanding, there are noticeable differences between the goals and methodologies of these two fields. Without attempting to characterize the differences in this short paper, we focus only on one crucial difference between them, and that is the prominent importance of *stability* in control engineering. This is understandable as many applications of control engineering are safety-critical. Proving that a controller (i.e., policy) makes the dynamical system stable, especially when there are uncertainties about the dynamics, requires making certain assumptions about the system. The challenge is to make as few assumptions as possible while still being able to guarantee the stability. On the other hand, methods in machine learning, in general, and reinforcement learning, in particular, are often capable of providing good solutions with minimal assumptions about the underlying system. Their guarantees, if there is any, is often in the form of suboptimality of the solution, which does not necessarily translate into the stability of the resulting policy.

We would like to have a controller design methodology that brings the best of these worlds together. Ideally, such a methodology can work with a large class of problems and can incorporate the prior knowledge about the system (which is often available in many engineering problems). The designed controller must be guaranteed to be stable, so it can be used in safety-critical problems. Its performance should also gradually improve while interacting with the environment. In this paper, we introduce such a method, which merges together a model-based controller and a data-driven machine learning algorithm. The model-based controller is designed to ensure stability, while the data-driven module improves the performance of the closed-loop system over time. More specifically, we use tools from indirect modular adaptive nonlinear control design and the Bayesian optimization technique of Gaussian Process-Upper Confidence Bound (GP-UCB) [Srinivas et al., 2010].

We take the indirect modular approach to adaptive nonlinear control design, e.g., Krstic et al. [1995]; Wang et al. [2006]; Benosman and Atinc [2013]; Atinc and Benosman [2013]; Benosman [2014]. In the indirect approach, a controller is designed by assuming that all the parameters of the dynamical system are known (certainly equivalence principle) while an estimator is used to estimate the unknown parameters online. When the design of the estimator is independent of the designed controller, the approach is called “modular”. The proposed modular adaptive control uses a robust model-based nonlinear controller to ensure the stability of the system—in the input-to-state stable (ISS) sense. The learning-based algorithm, GP-UCB in particular, estimates the uncertain parameters of the dynamical system in order to maximize the performance.

Our approach can be seen as solving a continuous-armed bandit problem in which each action corresponds to a set of parameters passed to a controller design mechanism. By defining the reward signal as the closed-loop performance of the controller for the selected action, we may use GP-UCB, a method for trading-off exploration-exploitation in the bandit setting, to orchestrate the selection of actions, and thus improving the closed-loop performance.

We would like to note that a modular design that combines model-based control and a model-free algorithm, in particular an extreme seeking-based (ES) algorithm, has been proposed before [Haghi and Ariyur, 2013; Benosman and Atinc, 2013; Atinc and Benosman, 2013; Benosman, 2014; Xia and Benosman, 2015]. The limitation of the ES-based algorithm is that it converges to a local minimum. GP-UCB, on the other hand, has a guarantee, in the form of a no-regret compared to the global optimum.

2 Problem Formulation

We consider a large class of affine uncertain nonlinear systems of the form

$$\begin{aligned} \dot{x} &= f(x) + \Delta f(t,x) + g(x)u, \\ y &= h(x), \end{aligned} \tag{1}$$

where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^p$, $y \in \mathbb{R}^m$ ($p \geq m$), represent the state, the input and the controlled output vectors, respectively. The vector field $\Delta f(t,x)$ represents the additive model uncertainties. The vector fields f, Δf, columns of g and function h satisfy the following assumptions.

Assumption A1

- The function $f : \mathbb{R}^n \to \mathbb{R}^n$ and the columns of $g : \mathbb{R}^n \to \mathbb{R}^p$ are $\mathbb{C}^\infty$ vector fields on a bounded set X of $\mathbb{R}^n$ and $h : \mathbb{R}^n \to \mathbb{R}^m$ is a $\mathbb{C}^\infty$ vector on X. The vector field $\Delta f(x)$ is $\mathbb{C}^1$ on X.
- System (1) has a well-defined (vector) relative degree $\{r_1,\ r_2,\ \cdots,\ r_m\}$ at each point $x^0 \in X$, and the system is linearizable, i.e., $\sum_{i=1}^m r_i = n$.
- The desired output trajectories y_{id} ($1 \leq i \leq m$) are C^∞ functions of time, relating desired initial points $y_{id}(0)$ at $t = 0$ to desired final points $y_{id}(t_f)$ at $t = t_f$.

The goal is to design a state feedback adaptive controller such that the output tracking error is uniformly bounded, with an upper bound that is a function of the estimation error of the uncertain parameters. Meanwhile, the estimation error is gradually being decreased by the model-free learning algorithm. Note that the goal of the learning algorithm is not stabilization but rather performance optimization, i.e., the learning improves the parameters' estimation error, which in turn improves the output tracking error. To achieve this control objective, we proceed as follows: First, we design a robust controller that guarantees input-to-state stability (ISS) of the tracking error dynamics w.r.t. the estimation errors input (Section 3). Then, we combine this controller with a model-free learning algorithm to iteratively estimate the uncertain parameters, by online optimizing a user-selected cost function (Section 4).

3 Controller Design

The controller design has two steps. First, we design a stabilizing controller under nominal conditions, i.e., when $\Delta f(t,x) = 0$. Then we design a robust controller that ensures that the tracking error is guaranteed to be bounded even when we have estimate error in the parameters of the system.

Under the nominal condition $\Delta f(t,x) = 0$, it is well know (cf. Khalil [2002]) that system (1) can be written as

$$y^{(r)}(t) \;=\; b(\xi(t)) + A(\xi(t))u(t), \tag{2}$$

where $y^{(r)}(t) = [y_1^{(r_1)}(t),\ y_2^{(r_2)}(t),\ \cdots,\ y_m^{(r_m)}(t)]^\top$, $\xi(t) = [\xi^1(t),\ \cdots,\ \xi^m(t)]^\top$, and $\xi^i(t) = [y_i(t),\ \cdots,\ y_i^{(r_i-1)}(t)]$ for $1 \le i \le m$. The functions $b(\xi)$ and $A(\xi)$ can be written as functions of f, g and h. $A(\xi)$ is non-singular in $\tilde{X}$, where $\tilde{X}$ is the image of the set of X by the diffeomorphism $x \mapsto \xi$ between the states of system (1) and the linearized model (2). To deal with the uncertain model, we need to introduce one more assumption on system (1).

Assumption A2 The additive uncertainties $\Delta f(t,x)$ in (1) appear as additive uncertainties in the input-output linearized model (2) as follows:

$$y^{(r)}(t) \;=\; b(\xi(t)) + A(\xi(t))u(t) + \Delta b(t,\xi(t)), \tag{3}$$

where $\Delta b(t,\xi)$ is $\mathbb{C}^1$ w.r.t. the state vector $\xi \in \tilde{X}$.

It is well-known that the nominal model (2) can be easily transformed into a linear input-output mapping. We can first define a virtual input vector $v(t)$ as

$$v(t) \;=\; b(\xi(t)) + A(\xi(t))u(t). \tag{4}$$

Combining (2) and (4), we obtain the following input-output mapping:

$$y^{(r)}(t) \;=\; v(t). \tag{5}$$

Based on the linear system (5), it is straightforward to design a stabilizing controller for the nominal system (2) as

$$u_n = A^{-1}(\xi)\left[v_s(t,\xi) - b(\xi)\right], \tag{6}$$

where v_s is a $m \times 1$ vector and the i-th ($1 \le i \le m$) element v_{si} is given by

$$v_{si} = y_{id}^{(r_i)} - K_{r_i}^i\left(y_i^{(r_i-1)} - y_{id}^{(r_i-1)}\right) - \cdots - K_1^i(y_i - y_{id}). \tag{7}$$

If we denote the tracking error as $e_i(t) \triangleq y_i(t) - y_{id}(t)$, we obtain the following tracking error dynamics

$$e_i^{(r_i)}(t) + K_{r_i}^i e^{(r_i-1)}(t) + \cdots + K_1^i e_i(t) = 0, \tag{8}$$

where $i \in \{1,2,\ldots,m\}$. By properly selecting the gains K_j^i ($i \in \{1,2,\ldots,m\}$ and $j \in \{1,2,\ldots,r_i\}$), which exist under certain assumptions, we can obtain global asymptotic stability of the tracking errors $e_i(t)$. Refer to [Benosman and Farahmand, 2016; Benosman et al., 2016] for the detail. It can be shown that there exists a positive definite matrix $P > 0$ such that (see e.g. Khalil [2002])

$$\tilde{A}^T P + P\tilde{A} \;=\; -\mathbf{I} \tag{9}$$

where $\tilde{A} \in \mathbb{R}^{n\times n}$ is a diagonal block matrix given by $\tilde{A} \;=\; \mathrm{diag}\{\tilde{A}_1, \tilde{A}_2, \ldots, \tilde{A}_m\}$, and $\tilde{A}_i$ ($1 \le i \le m$) is a $r_i \times r_i$ matrix given by

$$\tilde{A}_i \;=\; \begin{bmatrix} 0 & 1 & & & \\ 0 & 0 & 1 & & \\ \vdots & \vdots & \vdots & & 1 \\ -K_1^i & -K_2^i & \cdots & \cdots & -K_{r_i}^i \end{bmatrix}.$$

We now build a robust ISS controller for the uncertain model with $\Delta f(t,x) \neq 0$. The corresponding exact linearized model is given by (3) where $\Delta b(t,\xi(t)) \neq 0$. The global asymptotic stability of the error dynamics (8) cannot be guaranteed anymore due to the additive uncertainty $\Delta b(t,\xi(t))$. We use Lyapunov reconstruction techniques to design a new controller so that the tracking error is guaranteed to be bounded given that the estimate error of $\Delta b(t,\xi(t))$ is bounded.

The new controller for the uncertain model (3) is defined as $u_f = u_n + u_r$, where the nominal controller u_n is given by (6) and the robust controller u_r will be defined shortly. By plugging-in this controller in the dynamics (3), we obtain

$$\begin{aligned} y^{(r)}(t) &= b(\xi(t)) + A(\xi(t))u_f + \Delta b(t,\xi(t)) \;=\; b(\xi(t)) + A(\xi(t))u_n + A(\xi(t))u_r + \Delta b(t,\xi(t)) \\ &= v_s(t,\xi) + A(\xi(t))u_r + \Delta b(t,\xi(t)). \end{aligned} \tag{10}$$

This leads to the following error dynamics

$$\dot{z} \;=\; \tilde{A}z + \tilde{B}\delta, \tag{11}$$

where $\tilde{A}$ is defined as above (9), δ is a $m \times 1$ vector given by $\delta = A(\xi(t))u_r + \Delta b(t,\xi(t))$ and the matrix $\tilde{B} \in \mathbb{R}^{n\times m}$ is given by $\tilde{B}^\top = \left[\tilde{B}_1^\top \ldots \tilde{B}_m^\top\right]^\top$ where each $\tilde{B}_i$ $(1 \leq i \leq m)$ is given by a $r_i \times m$ matrix such that $\tilde{B}_i(l,q) = 1$ for $l = r_1$ and $q = i$, and 0 otherwise. If we choose $V(z) = z^\top P z$ as a Lyapunov function for the dynamics (11), where P is the solution of the Lyapunov equation (9), we obtain

$$\dot{V}(t) \;=\; \frac{\partial V}{\partial z}\dot{z} = \;=\; z^T(\tilde{A}^T P + P\tilde{A})z + 2z^T P\tilde{B}\delta = -\|z\|^2 + 2z^T P\tilde{B}\delta. \tag{12}$$

Next, we design the controller u_r based on the form of the uncertainties $\Delta b(t,\xi(t))$. More specifically, we consider here the case when $\Delta b(t,\xi(t))$ is of the following form

$$\Delta b(t,\xi(t)) \;=\; E\,Q(\xi,t), \tag{13}$$

where $E \in \mathbb{R}^{m\times m}$ is a matrix of unknown constant parameters, and $Q(\xi,t):\ \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^m$ is a known bounded function of state and time variables. We denote the estimate of E by $\hat{E}(t)$ and the estimate error by $e_E = E - \hat{E}$. We define the unknown parameter vector $\Delta = [E(1,1),...,E(m,m)]^T \in \mathbb{R}^{m^2}$, i.e., concatenation of all elements of E, its estimate is denoted by $\widehat{\Delta}(t) = [\hat{E}(1,1),...,\hat{E}(m,m)]^T$, and the estimation error vector is given by $e_\Delta(t) = \Delta - \widehat{\Delta}(t)$. We now define the following robust controller:

$$u_r \;=\; -\,A^{-1}(\xi)[\tilde{B}^T P z\|Q(\xi,t)\|^2 + \widehat{E}(t)Q(\xi,t)]. \tag{14}$$

The closed-loop error dynamics can be written as

$$\dot{z} \;=\; \tilde{f}(t,z,e_\Delta), \tag{15}$$

where $e_\Delta(t)$ is considered to be an input to system (15).

Theorem 1. *Consider the system (1) and assume that Assumptions A1, A2 hold, $\Delta b(t,\xi(t))$ satisfies (13), and the gains K_j^i stabilize (8). If we apply the feedback controller $u_f = u_n + u_r$ (cf. (6) and (14)) to the uncertain nonlinear dynamical system (1), the closed-loop system (15) is ISS from the estimation errors input $e_\Delta(t) \in \mathbb{R}^{m^2}$ to the tracking errors state $z(t) \in \mathbb{R}^n$.*

4 GP-UCB-based Uncertainties Estimation

The robust controller in the previous section ensures that the closed-loop system is ISS. The performance of the system, however, depends on the estimate $\widehat{\Delta}(t)$ used in the robust controller design (14). There are various ways to define the performance measure. For example, for a finite horizon $0 < T_f < \infty$, we may define it as

$$J(\widehat{\Delta}) = \int_0^{T_f} \left\|e(t;\widehat{\Delta})\right\|^2 + \left\|u_f(t;\widehat{\Delta})\right\|^2 \mathrm{d}t, \tag{16}$$

in which $e(t;\widehat{\Delta})$ and $u_f(t;\widehat{\Delta})$ are the tracking error and the control signal when $\widehat{\Delta}$ is used to design the controller.

To guide the search for $\widehat{\Delta}$ leading to a high-performance controller, we propose to use GP-UCB [Srinivas et al., 2010]. GP-UCB is a Bayesian approach for stochastic optimization, i.e., the task of finding the global optimum of an unknown function when the evaluations are potentially contaminated with noise. GP-UCB balances the exploration-exploitation in the continuous-armed bandit setting and has a guarantee in the form of cumulative regret bound.

We briefly describe GP-UCB following the original paper [Srinivas et al., 2010]. Consider the cost function $J: D \to \mathbb{R}$ to be minimized, e.g., (16). This function depends on the dynamics of the closed-loop system, which itself depends on the parameters $\widehat{\Delta}$ used in the controller design (14). A Gaussian Process (GP) [Rasmussen and Williams, 2006] is defined by its mean function $\mu(\widehat{\Delta}) = \mathbb{E}\left[J(\widehat{\Delta})\right]$ and covariance function (or kernel) $\kappa(\widehat{\Delta}_1, \widehat{\Delta}_2) = \mathrm{Cov}(J(\widehat{\Delta}_1), J(\widehat{\Delta}_2))$. An example of the kernel function is $\kappa(\widehat{\Delta}_1, \widehat{\Delta}_2) = \exp(-\frac{\|\widehat{\Delta}_1 - \widehat{\Delta}_2\|^2}{2l^2})$, the squared exponential kernel with length scale $l > 0$.

The combination of GP-UCB and the controller design method of Section 3 works iteratively. Suppose that we are at iteration $\tau = 1, 2, \dots$ and we have already selected the set of parameters $\underline{\widehat{\Delta}}_{\tau-1} \triangleq \{\widehat{\Delta}_1, \widehat{\Delta}_2, \dots, \widehat{\Delta}_{\tau-1}\} \subset D$. For each $i = 1, \dots, \tau - 1$, we have also observed the noisy evaluation $y_i = J(\widehat{\Delta}_i) + \eta_i$ with $\eta_i \sim N(0, \sigma^2)$ being i.i.d. Gaussian noise. The value of y_i is obtained after using $\widehat{\Delta}_i$ to design the robust controller (cf. (14)) and running the closed-loop dynamical system with the control signal $u_f = u_n + u_r$ (cf. (6) and (14)).

GP-UCB requires the computation of the posterior mean and variance of the GP given data. Suppose that initially we started with $\mathrm{GP}(0, \kappa)$, a GP with zero prior mean. We find the posterior mean and variance for a new point $\widehat{\Delta}^* \in D$ as follows: Denote the vector of observed values by $\mathbf{y}_{\tau-1} = [y_1, \dots, y_{\tau-1}]^\top \in \mathbb{R}^{\tau-1}$, and define the Grammian matrix $K \in \mathbb{R}^{\tau-1\times\tau-1}$ with $[K]_{i,j} = \kappa(\widehat{\Delta}_i, \widehat{\Delta}_j)$, and the vector $\kappa_* = [\kappa(\underline{\widehat{\Delta}}_1, \widehat{\Delta}^*), \dots, \kappa(\underline{\widehat{\Delta}}_{\tau-1}, \widehat{\Delta}^*)]$. The expected mean $\mu_{\tau-1}(\widehat{\Delta}^*)$ and the variance $\sigma_{\tau-1}(\widehat{\Delta}^*)$ of the posterior of the GP evaluated at $\widehat{\Delta}^*$ are (cf. Section 2.2 of Rasmussen and Williams [2006]) $\mu_{\tau-1}(\widehat{\Delta}^*) = \kappa_* \left[K + \sigma^2\mathbf{I}\right]^{-1} \mathbf{y}_{\tau-1}$ and $\sigma^2_{\tau-1}(\widehat{\Delta}^*) = \kappa(\widehat{\Delta}^*, \widehat{\Delta}^*) - \kappa_*^\top \left[K + \sigma^2\mathbf{I}\right]^{-1} \kappa_*$. At iteration τ, the GP-UCB algorithm selects the next query point $\widehat{\Delta}_\tau$ by solving the following optimization problem:

$$\widehat{\Delta}_\tau \leftarrow \operatorname*{argmin}_{\widehat{\Delta} \in D} \mu_{\tau-1}(\widehat{\Delta}) - \beta_\tau^{1/2} \sigma_{\tau-1}(\widehat{\Delta}). \tag{17}$$

Here β_τ depends on the choice of kernel among other parameters of the problem [Srinivas et al., 2010]. This process repeats. The optimization problem (17) is often nonlinear and non-convex. Nonetheless, solving it only requires querying the GP, which in general is much faster than querying the original dynamical system. This is important when the dynamical system is a physical system and having as few number of interactions as possible is crucial.

5 Conclusion

We used tools from machine learning and modern control engineering to design an adaptive robust controller for a class of nonlinear systems. The designed controller is guaranteed to be input-state stable, while the Bayesian optimization technique allows its performance to improve. We have empirically studied this approach for the control of a two-link manipulator with favourable results [Benosman and Farahmand, 2016]. One of the main advantages of the proposed controller, compared to the existing model-based adaptive controllers, is that we can estimate multiple uncertainties at the same time even if they appear in the model equation in a challenging structure, e.g., linearly dependent uncertainties affecting only one output, or uncertainties appearing in a nonlinear term of the model, which are well-known limitations of the model-based estimation approaches.

References

G. Atinc and M. Benosman. Nonlinear learning-based adaptive control for electromagnetic actuators with proof of stability. In *IEEE, Conference on Decision and Control*, pages 1277–1282, 2013.

M. Benosman. Extremum-seeking based adaptive control for nonlinear systems. In *IFAC World Congress*, pages 401–406, 2014.

M. Benosman and G. Atinc. Multi-parametric extremum seeking-based learning control for electromagnetic actuators. In *American Control Conference*, pages 1914–1919, 2013.

M. Benosman and A.-m Farahmand. Bayesian optimization-based modular indirect adaptive control for a class of nonlinear systems. In *IFAC International Workshop on Adaptation and Learning in Control and Signal Processing*, 2016.

M. Benosman, A.-m Farahmand, and M. Xia. Learning-based modular indirect adaptive control for a class of nonlinear systems. In *American Control Conference (ACC)*, 2016.

P. Haghi and K. Ariyur. Adpative feedback linearization of nonlinear MIMO systems using ES-MRAC. In *American Control Conference*, pages 1828–1833, 2013.

H. Khalil. *Nonlinear Systems*. Prentice Hall, 3rd edition, 2002.

M. Krstic, I. Kanellakopoulos, and P. Kokotovic. *Nonlinear and Adpative Control Design*. Wiley, New York, 1995.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

N. Srinivas, A. Krause, S. M Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 1015–1022, 2010.

C. Wang, D. J Hill, S. S. Ge, and G. Chen. An ISS-modular approach for adaptive neural control of pure-feedback systems. *Automatica*, 42(5):723–731, 2006.

M. Xia and M. Benosman. Extremum seeking-based indirect adaptive control for nonlinear systems with time-varying uncertainties. In *European Control Conference*, pages 2780–2785, 2015.

Optimal Continuous State Planning with Semantic Observations

Luke Burks
Smead Aerospace Engineering Sciences
University of Colorado Boulder
`luke.burks@colorado.edu`

Nisar Ahmed
Smead Aerospace Engineering Sciences
University of Colorado Boulder
`nisar.ahmed@colorado.edu`

Abstract

Many applications of planning under uncertainty require autonomous agents to reason over outcomes in continuous dynamical environments using imprecise but readily available semantic observations. For instance, in extended search and tracking applications, small autonomous unmanned aircraft must be able to efficiently reacquire and localize mobile targets that can potentially remain out of view for long periods of time; planning algorithms must generate vehicle trajectories that optimally exploit imperfect detection data from onboard sensors, as well as semantic natural language observations that can be opportunistically provided by human supervisors. This work develops novel strategies for optimal planning with semantic observations using continuous state Partially Observable Markov Decision Processes (CPOMDPs). We propose two major innovations to Gaussian mixture (GM) CPOMDP policy approximation methods. Our innovations address the fact that, while these state of the art methods have many theoretically nice properties, they are hampered by the inability to efficiently represent and reason over hybrid continuous-discrete probabilistic models. Firstly, closed-form variational GM approximations of PBVI Bellman policy backups are derived using softmax models of continuous-discrete semantic observation probabilities. Secondly, a new clustering-based technique for condensation of GMs is introduced for efficient scaling to large GMs. We show that GM policies resulting from our proposed methods result in policies that are as effective as those produced by other state of the art GM approximation approaches, although our methods require significantly less modeling overhead and runtime cost. We show results for a combined localization and target search task based on semantic binary observations.

Keywords: Continuous state POMDPs, variational Bayes, Gaussian mixtures, softmax models, semantic reasoning, hybrid inference

Acknowledgements

This work was supported by the NSF IUCRC Center for Unmanned Aircraft Systems (C-UAS) project CU14-02, ”Harnessing Human Perception for UAS via Bayesian Active Sensing”.

1 Introduction

In small unmanned aircraft systems (UAS), human operators and users can play valuable roles as 'human sensors' that contribute information beyond the reach of autonomous air vehicle sensors. For instance, operators in search and tracking missions can provide 'soft data' to narrow down possible survivor locations using semantic natural language observations (e.g. 'Nothing is around the lake'; 'Something is moving towards the fence'), or provide estimates of physical quantities (e.g. masses/sizes of obstacles, distances from landmarks) to help autonomous vehicles better understand search areas and improve online decision making with limited computational resources. This research focuses on the development of intelligent operator-UAS interfaces that not only leverage combined robotic sensing and semantic human sensing, but are also tightly integrated with vehicle motion planning in large continuous dynamic spaces. Previous work has looked at modeling semantic natural language data from human sensors in continuous dynamic state spaces using softmax observation likelihoods [2, 8], and explored the use of convolutional neural networks (CNNs) to compute the Value of Information (VOI) for a given human sensor query [5]. However, the problem of integrating semantic sensing with planning and decision-making in continuous spaces remains fairly open. While such problems are most naturally formulated as continuous state POMDPs (CPOMDPs), several important limitations must still be addressed for existing CPOMDP solution strategies to be practically viable and deployable for real systems. To this end, this work presents a novel variational Bayes (VB) Gaussian mixture (GM) approximation method for finding CPOMDP policies based on semantic sensor observations that are modeled by softmax likelihood models (which can be used to model data provided by autonomous detection algorithms as well as human sensors). We also propose a novel technique for quickly and efficiently condensing GM CPOMDP policy approximations, allowing these techniques to scale to more complex problems and larger state spaces. Finally, we provide preliminary results of our proposed VB CPOMDP policy approximation on a simple dynamic target search problem, showing favorable comparisons to the existing state-of-the-art approximation method of [4].

2 Problem Definition and Continuous State POMDP Preliminaries

Formally, a POMDP is described by the 7-tuple $(S, A, T, R, \Omega, O, \gamma)$, where: S is a set of states s; A is a set of actions a; T is a discrete time probabilistic transition mapping from state s to state s' given some a; R is the immediate reward mapping over (s, a) pairs; Ω is a set of observations o; O is the likelihood mapping from states to observations; and $\gamma \in [0, 1]$ is a discount factor. An agent whose decision making process is modeled by a POMDP seeks to maximize a utility function defined by the expected future discounted reward: $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$, where $s_t \in S$ is the state at discrete time t, and $a_t \in A$. The expectation operator $\mathbb{E}\left[\cdot\right]$ reflects that the agent lacks full knowledge of s_t. It must instead rely on the noisy process model T and observation model O to update a Bayesian belief function $b(s_t) = p(s_t|a_{1:t}, o_{1:t})$, which summarizes all available information for reasoning about present and possible future states. A decision making policy $\pi(b(s_t)) \rightarrow a_t$ must therefore be found for any possible belief $b(s_t)$. Since POMDPs are equivalent to MDPs over $b(s_t)$, exact policies are impossible to compute for all but the simplest problems.

One family of well-known techniques for computing approximate POMDP policies offline is Point-Based Value Iteration (PBVI) [6]. These methods approximate π at a finite set of 'typical' sample beliefs $b(s)$, for which explicit finite-horizon Bellman equation recursions can be performed to obtain locally optimal actions. When S is a set of discrete states with N possible outcomes, then $b(s) \in \mathbb{R}^N$ such that $\sum_{s=1}^{N} b(s) = 1$. In this case, PBVI policies are often represented by a set Γ of vectors $\alpha \in \mathbb{R}^N$ for each possible action $a \in A$. The α vectors encode the value function over the state space as linear hyperplanes. The action a recommended by the policy for a given $b(s) \in \mathbb{R}^N$ is found as $\arg\max_{\alpha^a \in \Gamma} < \alpha, b(s) >$, where $< \cdot >$ is the inner product and α^a is an α vector associated to action $a \in A$. A number of methods exist for sampling beliefs, e.g. starting with a large randomly collected set of $b(s)$ elements [6] (the approach used in this work), or starting with a smaller set and propagating them between recursive Bellman updates for α vector computations. When s is a continuous random vector such that $s \in \mathbb{R}^N$ with support $\mathcal{S}(S)$, it is more natural to represent $b(s)$ as a probability density function (pdf), where $\int_{\mathcal{S}(S)} b(s)ds$. In such cases, CPOMDPs can be formulated by specifying T, R, O and $\alpha(s)$ to be suitable continuous functions over s. Although $b(s)$ can sometimes be represented by simple parametric models such as Gaussian pdfs [1], $b(s)$ is in general analytically intractable for arbitrary T and O models (e.g. nonlinear dynamics, semantic sensing). Therefore, $b(s)$ must also be approximated to derive a suitable set Γ of approximate functions $\alpha(s)$, such that actions for the (approximate) policy $\pi(b(s))$ are given by $\arg\max < \alpha(s), b(s) >$.

2.1 Gaussian Mixture CPOMDPs

Finite Gaussian mixture (GM) models provide a very general and flexible way to approximate arbitrary functions $f(s)$ of interest for CPOMDPs, where

$$f(s) = \sum_{m=1}^{M} w_m \phi(s|\mu_m, \Sigma_m)$$

is a GM defined by M weights $w_m \in \mathbb{R}_{\geq 0}$, means $\mu_m \in \mathbb{R}^N$, and covariance matrices $\Sigma_m \in \mathbb{R}^{N \times N}$ for the multivariate normal component pdf $\phi(s|\mu_m, \Sigma_m)$, such that $\sum_{m=1}^{M} w_m = 1$ to ensure normalization when $f(\cdot)$ represents a pdf (this condition need not apply otherwise). Ref. [4] showed that if A describes a discrete action space and $T = p(s'|s,a)$, $O = p(o|s')$, and $R = r_a(s)$ are all specified by finite GM functions over s, then PBVI approximations to $\pi(b(s))$ can be found based on closed-form GM recursions for $b(s)$ and $\alpha(s)$, where

$$b(s) = \sum_{j}^{J} w_j \phi(s|\mu_j, \Sigma_j), \quad \alpha_n = \sum_{k=1}^{M} w_k \phi(s|\mu_k, \Sigma_k),$$

$$< \alpha, b > = \int_s \left[\sum_{k}^{M} w_k \phi(s|\mu_k, \Sigma_k) \right] \left[\sum_{j}^{J} w_j \phi(s|\mu_j, \Sigma_j) \right] ds = \sum_{k,j}^{M \times J} w_k w_j \phi(\mu_j|\mu_k, \Sigma_j + \Sigma_k),$$

and the α functions are recursively computed in closed-form via finite horizon PBVI Bellman backups for time steps $0, ..., n$,

$$\alpha^i_{a,o}(s) = \int_{s'} \alpha^j_{n-1}(s') p(o|s') p(s'|s,a) ds', \qquad \alpha^i_n = r_a(s) + \gamma \sum_o \arg\max_{\alpha^j_{a,o}} (< \alpha^j_{a,o}, b >).$$

2.2 Limitations for Hybrid Continuous-Discrete Reasoning and Example Semantic Sensing Search Problem

If $o \in \Omega$ describes a categorical/discrete-valued semantic observation with $N_o = |\Omega|$ possible values, then the observation likelihood function $O = p(o|s)$ must describe a valid hybrid (continuous-discrete) probability distribution, such that $\sum_o p(o|s) = 1 \; \forall s \in \mathcal{S}(S)$. The current state-of-the-art is to model O by an unnormalized GM for each possible outcome o [4], $p(o|s) \approx \sum_{l_o=1}^{L_o} w_o \phi(s|\mu_{l_o}, \Sigma_{l_o})$, such that $\sum_o p(o|s) \approx 1$ everywhere. Although this preserves the closed-form updates required for PBVI, such models are often very difficult and labor intensive to specify. In particular, for $N \geq 2$, L_o must be very large for each possible o to ensure that the normalization requirement is satisfied for all s and that desired probabilities in $p(o|s)$ are modeled accurately. This effectively turns $p(o|s)$ into a 'soft discretization' model based on GMs and severely restricts the scalability of GM policy approximation. Another related and more general problem is the fact that the GM multiplication and summation operations in the α recursions defined above each lead to a drastic increase in the number of resulting GM components. GM condensation methods are thus needed to control the size of $b(s)$ and $\alpha^a(s)$. However, these can be computationally expensive and introduce additional approximations. The use of dense unnormalized GM models for semantic likelihoods O exacerbates this issue and introduces additional errors if normalization is not guaranteed for all $s \in \mathcal{S}(S)$.

For concreteness, consider a simple $N = 2$ CPOMDP in which an autonomous robot 'cop' attempts to localize and catch a mobile 'robber', where both are constrained to move along parallel linear paths (see Figure 1). Here, $S = \mathbb{R} \times \mathbb{R}$ consists of two bounded continuous random variables at each discrete time step t, $s = [Cop, Rob]^T$, $Cop \in (0,5)$, $Rob \in (0,5)$. The robber executes a Gaussian random walk: $p(Rob_{t+1} = \phi(Rob_{t+1}|Rob_t, 0.5)$. The cop must choose from among 3 noisy actions $A = \{\text{left, right, stay}\}$ to define a movement direction, such that: $p(Cop_{t+1}|Cop_t, \text{left}) = \phi(Cop_{t+1}|Cop_t - 0.5, 0.01)$, $p(Cop_{t+1}|Cop_t, \text{right}) = \phi(Cop_{t+1}|Cop_t + 0.5, 0.01)$, and $p(Cop_{t+1}|Cop_t = Cop_{t+1}, \text{stay}) = 1$. The reward function is defined to reward the cop for remaining within a set distance of the robber's position, and to penalize otherwise,

$$r(|Rob_t - Cop_t| <= 0.5) = 3, \quad r(|Rob_t - Cop_t| > 0.5) = -1.$$

The cop obtains simple binary semantic observations o_t from a noisy sensor (e.g. human supervisor or onboard visual detector), where $o_t \in \{\text{'robber detected'}, \text{'robber not detected'}\}$. Figure 2 (a) shows GM models for the 'detection' and 'no detection' likelihoods, which are respectively parameterized by 8 and 200 isotropic Gaussian components, for 624 parameters total.

3 Variational CPOMDP Approximations for Softmax Semantic Observation Likelihoods

Semantic observation likelihoods should ideally be modeled by self-normalizing functions like the softmax model,

$$p(o|s) = \frac{\exp(w_o^T s + b_o)}{\sum_{c=1}^{N_o} \exp(w_c^T s + b_c)}$$

where $w_1, ..., w_{N_o} \in \mathbb{R}^N$ and $b_1, ..., b_{N_o}$ are the vector weight parameters and scalar bias parameters for each categorical outcome o given s. In addition to ensuring $\sum_o p(o|s) = 1 \; \forall s \in \mathcal{S}(S)$, softmax functions require relatively few parameters compared to GM likelihoods, and scale well to higher dimensional spaces. Figure 2 (b) shows how the cop's semantic observation likelihood can be easily modeled with a softmax function featuring 3 semantic categorical classes (two of which

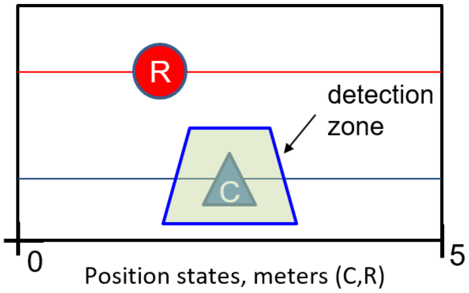
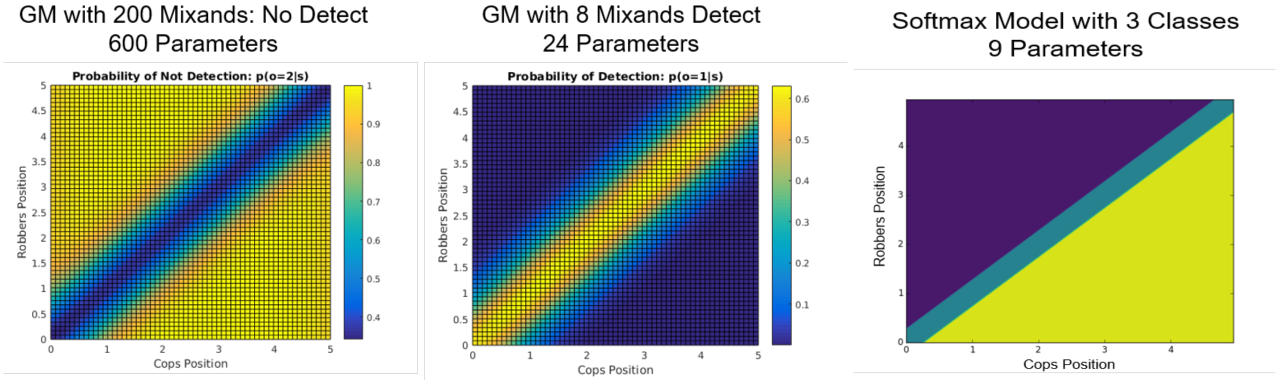


Figure 1: Set up for 'Cop and Robot' semantic target search problem.



(a) Gaussian Mixture Likelihood Function (b) Softmax Likelihood Function

Figure 2: Observation models compared by number of parameters.

collectively represent the 'no detect' observation in the blue and yellow regions via the 'multimodal softmax'/MMS formulation [8]). Unlike the GM likelihood function approxmation, the softmax model only requires 9 parameters. In general, softmax parameters can be easily synthesized to conform to a priori sensing geometry information and quickly calibrated/tuned with training data [8]. However, the product of GMs and softmax functions is analytically irreducible, which breaks the recursive nature of the α function updates for GM-based CPOMDPs. Thus, in order to use softmax models for the observation likelihoods $p(o|s)$ in the GM-based PBVI CPOMDP policy approximation described earlier, the local variational Bayes (VB) approximation for hybrid inference with softmax models developed in [2] is used to approximate the product of a softmax model and a GM as a variational GM,

$$\alpha^i_{n-1}(s')p(o|s') = \left[\sum_{k=1}^{M} w_k\phi(s|\mu_k,\Sigma_k)\right]\left[\frac{\exp w_o^T s'}{\sum_{c=1}^{S}\exp w_c^T s'}\right] \approx \sum_{v=1}^{V} w_v\phi(s'|\mu_v,\Sigma_v).$$

Figure 3 (left) shows the key idea behind the VB approximation: the softmax function (shown in blue) is approximated by a variational Gaussian lower bounding function (black) that ensures the product with another Gaussian function (green) results in a good Gaussian approximation (red dots) to the true non-Gaussian but unimodal product of the softmax and Gaussian functions (solid magenta). This allows us to reform the previous α update equation for PBVI as:

$$\alpha^i_{a,o}(s) = \int_{s'} \alpha^i_{n-1}(s')p(o|s')p(s'|s,a)ds' \approx \int_{s'}\left[\sum_{k} w^j_k\phi(s'|s^j_k,\Sigma^j_k)\right]\left[\frac{\exp w_o^T s'}{\sum_{c=1}^{S}\exp w_c^T s'}\right][\phi(s'|s+\Delta(a),\Sigma^a)]\,ds',$$

$$\approx \sum_{h=1}^{K} w_h\phi(s|\hat{\mu}_h-\Delta(a),\hat{\Sigma}_h+\Sigma^a) \quad (\text{where } (\Delta(a),\Sigma^a) = \text{known constants})$$

Recrusive semantic observation updates to GM $b(s)$ pdfs can also be carried out online during execution of these policies using softmax likelihoods, as shown in Figure 3 (right),

$$b(s') \propto p(o|s')\int_{s} p(s'|s,a)b(s)ds = \left(\frac{\exp w_r^T s'}{\sum_{c=1}^{S}\exp w_c^T s'}\right)\left[\sum_{j} w_j\phi(s'|s_j+\Delta(a),\Sigma_j+\Sigma^a)\right] \approx \sum_{z=1}^{Z} w_z\phi(s'|\mu_z,\Sigma_z).$$

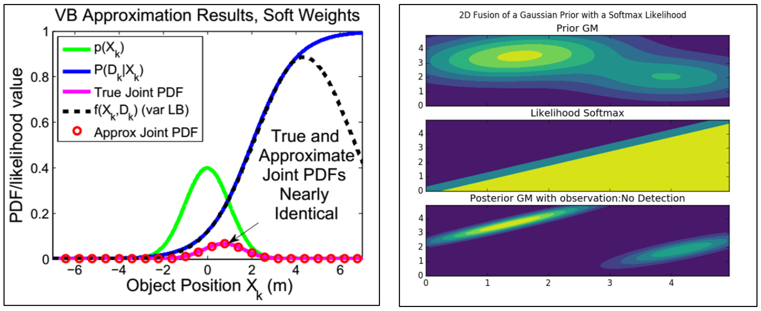
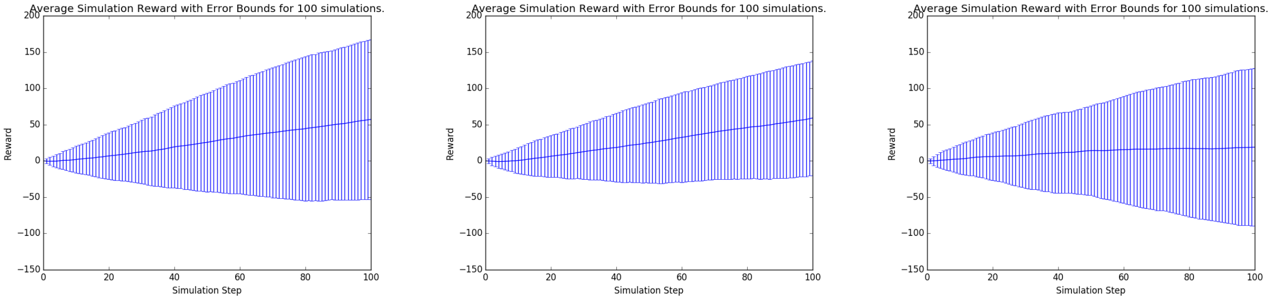


Figure 3: Variational Bayes approximation (left) and GM belief update (right) examples.



(a) VB-POMDP Average Final Reward: 57 (b) GM-POMDP Average Final Reward: 59 (c) Greedy Average Final Reward: 19

Figure 4: Rewards achieved on simple target search problem (2σ bounds for 100 simulation runs shown).

Existing GM condensation algorithms perform myopic pairwise merging of the M components in α_n^i, such that the resulting M' components in $\hat{\alpha_n^i}$ minimize some information loss metric [4, 3]. Naïve pairwise merging tends to be very expensive and slow when $M \geq 100$ (often the case for long horizon Bellman recursions with $N \geq 2$). To improve the speed of condensation, we employ a 'divide and conquer' strategy which first pre-classifies the mixture indices into local clusters (submixtures) using K-means, and then condenses each cluster to some pre-determined number of components via pairwise merging, before recombining the results to a condensed mixture with the desired size $M' < M$. For merging within submixture clusters, we use the Runnall's algorithm [7], which uses an upper bound on the KL divergence between the pre-merge and post-merge submixture to select the least dissimilar component pairs merging. Preliminary results indicate that our hybrid method achieves approximately the same accuracy (measured by the integral square difference metric for GMs [9]) for condensation performance as classical full scale pairwise merging, although our hybrid method is considerably cheaper and faster (e.g. 512 secs vs. 86.5 secs for $M = 2000 \rightarrow M' = 10$ with $N = 2$).

Fig. 4 compares the resulting policy approximations for the cop-robot search problem using the proposed VB-POMDP method (using the softmax likelihood model shown in Fig. 2b) and the GM-POMDP policy approximation of [4] (using the GM observation models shown in Fig. 2a); for reference on the optimality of both methods, results for a third greedy one-step implementation of the latter approximation is also shown. The proposed VB-POMDP policy approximation achieves a similar optimal performance to the baseline GM-POMDP policy, which is more expensive to generate.

References

[1] P Abbeel. Reinforcement learning for nonlinear dynamical systems and Gaussian belief space planning. *Reinforcement Learning and Decision Making*, page 13, 2013.

[2] N.. Ahmed, E. Sample, and M. Campbell. Bayesian multicategorical soft data fusion for human-robot collaboration. *IEEE Trans. on Robotics*, 29(1):189–206, 2013.

[3] Emma Brunskill, Leslie Pack Kaelbling, Tomas Lozano-Perez, and Nicholas Roy. Planning in partially-observable switching-mode continuous domains. *Annals of Mathematics and Artificial Intelligence*, 58(3):185–216, 2010.

[4] M Spaan P Poupart JM Porta, N Vlassis. Point-based value iteration for continuous POMDPs. *IJCAI Int'l Joint Conference on Artificial Intelligence*, 7:1968–1974, 2011.

[5] K. Lore, N. Sweet, and N. Ahmed. Deep value of information estimators for collaborative human-machine information gathering. *Int'l Conference on Cyber-Physical Systems*, 2016.

[6] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. *IJCAI International Joint Conference on Artificial Intelligence*, pages 1025–1030, 2003.

[7] Andrew R. Runnalls. Kullback-Leibler approach to Gaussian mixture reduction. *IEEE Transactions on Aerospace and Electronic Systems*, 43(3):989–999, 2007.

[8] Nicholas Sweet and Nisar Ahmed. Structured synthesis and compression of semantic human sensor models for Bayesian estimation. *Proceedings of the American Control Conference*, 2016-July(2):5479–5485, 2016.

[9] Jason L. Williams and Peter S. Maybeck. Cost-function-based Gaussian mixture reduction for target tracking. *Proceedings of the 6th International Conference on Information Fusion, FUSION 2003*, 2:1047–1054, 2003.

Importance Sampling for Fair Policy Selection

Shayan Doroudi
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
`shayand@cs.cmu.edu`

Philip S. Thomas
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
`philipt@cs.cmu.edu`

Emma Brunskill
Computer Science Department
Stanford University
Stanford, CA 94305
`ebrun@stanford.edu`

Abstract

We consider the problem of *off-policy policy selection* in reinforcement learning settings: using historical data generated from running some policy to compare a set of two or more new policies. Policy selection methods can be used, for example, to decide which policy should be deployed next when two or more batch reinforcement learning algorithms suggest different policies or when we want to compare a policy derived from data to a policy constructed by an expert. We show that existing approaches to policy selection based on importance sampling can be *unfair*: they can select the worse of two policies more often than not. We present two illustrative examples to show that this unfairness can adversely impact policy selection scenarios that may arise in practical settings. We then give sufficient conditions for when we can apply existing techniques to do policy selection fairly. Our hope is that this work will lead to more researchers thinking about the problems that arise in off-policy policy selection and how we may mitigate these problems, which we believe has been largely ignored in the literature.

Keywords: policy selection, policy evaluation, importance sampling

Acknowledgements

The research reported here was supported, in whole or in part, by the Institute of Education Sciences, U.S. Department of Education, through Grants R305A130215 and R305B150008 to Carnegie Mellon University. The opinions expressed are those of the authors and do not represent views of the Institute or the U.S. Dept. of Education.

1 Introduction

In this paper, we consider the problem of *off-policy policy selection* in reinforcement learning settings: using historical data generated from running some policy to compare a set of two or more new policies. Policy selection methods can be used, for example, to decide which policy should be deployed next when two or more batch reinforcement learning algorithms suggest different policies or when we want to compare a policy derived from data to a policy constructed by an expert. Importance sampling, a technique for predicting the performance of one policy using data generated from running a different policy [4], is at the foundation of many policy selection and policy search algorithms [3, 1, 2, 5, 6]. In this paper, we introduce the notion of fairness for policy selection algorithms, which we believe has not been considered in prior work. In the case of comparing two policies, we say that a policy selection algorithm is *fair* if it selects the better of the two policies more often than it selects the worse of the two policies. The primary contribution of this paper is that we show that standard policy selection algorithms based on importance sampling are often unfair. We illustrate this with two concrete examples of settings that may arise in practice. We then present sufficient conditions for when we can use importance sampling to make fair comparisons, which is a first step towards fair policy selection.

2 Background

2.1 Reinforcement Learning

We consider sequential decision making settings in stochastic domains. In such domains, an agent interacts with the environment, and in doing so, it generates a trajectory, $\tau \triangleq (O_0, A_1, R_1, O_1, A_2, R_2, \ldots, A_T, R_T, O_T)$, which is a sequence of observations, actions, and rewards, with trajectory length T. The observations and rewards are generated by the environment according to a stochastic process that is unknown. The agent chooses actions according to a stochastic policy π, which is a conditional probability distribution over actions A_t given the partial trajectory $\tau_{1:t-1} \triangleq (O_1, A_1, R_1, O_2, A_2, R_2, \ldots, O_{t-1})$ of prior observations, actions, and rewards. The value of a policy π, V^π, is the expected sum of rewards when the policy is used:

$$V^\pi \triangleq \mathbb{E}\left[\sum_{t=1}^{T} R_t \middle| \tau \sim \pi\right]$$

The agent's goal is to find and execute a policy with a large value.

In this paper, we consider offline (batch) reinforcement learning where we have a batch of data, called historical data, that was generated from some known behavior policy π_b. We are interested in doing **batch off-policy policy selection**: identifying a good policy for use in the future based on estimating its performance using the data from π_b. This typically involves policy estimation or evaluation of a policy π_e. If $\pi_e = \pi_b$ this is known as **on-policy policy evaluation**. Otherwise it is known as **off-policy policy evaluation**.

2.2 Importance Sampling

In this paper, we focus on estimators that use importance sampling for off-policy policy selection. Model-based off-policy estimators tend to have lower variance than importance-sampling-based estimators, but at the cost of being biased and asymptotically incorrect (not consistent estimators of V^π) [3]. In contrast, importance sampling-based estimators can provide unbiased estimates of the value of a policy.

Suppose we have a batch of trajectories $\tau_1, \tau_2, \ldots, \tau_n$ sampled independently from executing a behavior policy π_b, but we want to estimate the value of another policy π_e. We can use the **importance sampling (IS) estimator** [4], which is given by:

$$\hat{V}_{\text{IS}}^{\pi_e} \triangleq \frac{1}{n}\sum_{i=1}^{n} w_i \sum_{t=1}^{T_i} R_{i,t}$$

where

$$w_i = \frac{\prod_{t=1}^{T_i} \pi_e(a_{i,t}|\tau_{i,1:t-1})}{\prod_{t=1}^{T_i} \pi_b(a_{i,t}|\tau_{i,1:t-1})}$$

The IS estimator is an unbiased and strongly consistent estimator of V^{π_e} if $\pi_e(a|\tau_{1:t-1}) = 0$ for all actions, a, and partial trajectories, $\tau_{1:t-1}$, where $\pi_b(a_t|\tau_{1:t-1}) = 0$. However, the IS estimator often has very large variance (which is at the root of why it can be unfair for policy selection, as we will show below). The **weighted importance sampling (WIS) estimator** is another estimator where instead of dividing the sum of the IS estimates for each trajectory by the number of trajectories, we divide by the sum of the importance weights as follows:

$$\hat{V}_{\text{WIS}}^{\pi_e} \triangleq \frac{1}{\sum_{i=1}^{n} w_i}\sum_{i=1}^{n} w_i \sum_{t=1}^{T_i} R_{i,t}$$

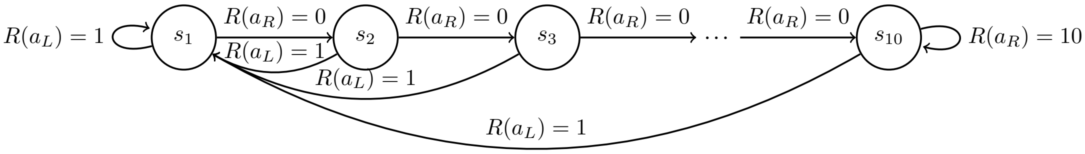


Figure 1: Domain in Section 3.1. The agent is in a chain of length 10. In each state, the agent can either go right (a_R) which progresses the agent along the chain and gives a reward of 0 unless the agent is in $s_1$0, in which case it gives a reward of 10 (and keeps the agent in the s_{10}), or go left (a_L), which takes the agent back to state s_1 and gives a reward of 1.

This estimator has less variance than the importance sampling estimator, but at the expense of adding some bias.

2.3 Policy Selection

A **policy selection** algorithm is any algorithm that takes as input an arbitrary number of policies, and outputs one of those policies. Any estimator used for policy evaluation can be transformed into a policy selection algorithm by simply evaluating each input policy and selecting the one that performs best under the estimator. Because policy evaluation is often used to do policy selection, the problem of policy selection has not been adequately studied independent of policy evaluation to our knowledge, even though it is perhaps the more important of the two problems since policy selection underlies the decision of what policy to use in practice. There are at least two properties that are desirable to have in a policy selection algorithm:

- Consistency: In the limit as the number of trajectories of historical data goes to infinity, the algorithm should always select the policy that has the largest value.
- Fairness: With *any* amount of data, the probability that the algorithm selects a policy with the largest value should be greater than the probability that it selects a policy that does not have the largest value. When choosing between two policies, this implies that the algorithm should choose the better policy at least half the time.

Since model-based approaches to policy evaluation are biased when the model class is inaccurate, they also do not satisfy these properties in general. For example, comparing the estimated value of the optimal policy from a set of models is both inconsistent and unfair (as even in the limit of infinite data, it may *always* pick the wrong policy) [3]. Importance sampling on the other hand, is consistent when used for policy selection as it is an unbiased and consistent estimator of the value function (so in the limit of infinite data, using IS will always lead to choosing the better policy); however we now show that it is not a fair policy selection algorithm.

3 Unfairness of Importance Sampling

We will give two examples that show the unfairness of importance sampling and how they can arise in counterintuitive ways in practically interesting settings, motivating why we should care about satisfying fairness.

3.1 Example 1: Bias Towards Myopic Policies

In this example, we show that using IS for policy selection could be biased in favor of myopic policies, which could be of great practical concern. This may come up in practical settings where we are interested in comparing more heuristic methods of planning (e.g., short look-ahead) to full-horizon planning methods. If we have the correct model class, full horizon planning is expected to be optimal, however it is both computationally expensive (so possibly not even tractable) and potentially sub-optimal if our model class is incorrect (e.g., our state representation is inaccurate or the world is a partially observable Markov decision process but we are modeling it as a fully observable Markov decision process). Thus, we may be interested in comparing full-horizon planning (or an approximation thereof) to myopic planning, and the following example shows that IS can sometimes favor policies resulting from myopic planning.

Consider the domain given in Figure 1. Now suppose we have data collected from a behavior policy π_b that takes each action with probability 0.5 and all trajectories have length 200. We want to compare two policies: π_{myopic} which takes a_L with probability 0.99 and a_R with probability 0.01, and π_{opt} which takes a_L with probability 0.01 and a_R with probability 0.99. (Note: the actual optimal policy is to always take a_R, for which π_{opt} is a slightly stochastic version.) Notice that the probability distribution of importance weights is the same for both π_{myopic} and π_{opt}, so both are equally close to the behavior policy in terms of probabilities over trajectories. However, for datasets that are not large enough, the importance sampling estimate will be larger for π_{myopic} than for π_{opt}, even though it is clearly the worse policy. In particular, when we have 1000 samples, **(1)** around 60% of the time, the importance sampling estimate of π_{myopic} is larger than that of π_{opt},

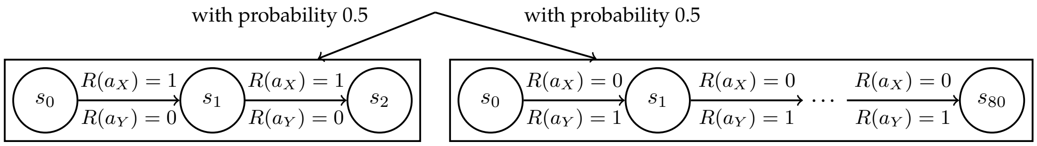


Figure 2: Domain in Section 3.2. The agent is placed uniformly at random in either a chain of length 2 or a chain of length L. At each time step, action a_X deterministically gives a reward of 1 to the agent if the agent is in the chain of length 2 and 0 otherwise, and action a_Y deterministically gives a reward of 1 to the agent if the agent is in the chain of length L and 0 otherwise. Both actions progress the agent along the chain.

	$\hat{V}_{\text{MC}}$	$\hat{V}_{\text{IS}}$	$\hat{V}_{\text{WIS}}$
π_X	1.39	0.98	1.98
π_Y	39.52	0.010	0.020

Table 1: Median estimates, out of 100 simulations, of different estimators using 100 samples of π_X and π_Y in the domain in Section 3.2.

and **(2)** around 95% of the time, the weighted importance sampling estimate of π_{myopic} is larger than that of π_{opt}. Thus both the IS and WIS estimators are unfair for policy selection.

The reason IS is unfair in this case is because one policy only gives high rewards in events that are unlikely under the behavior policy, and hence the behavior policy often does not see the high rewards of this policy as compared to a myopic policy. However, note that these events are still likely enough that we can build a model that would suggest choosing the optimal policy. IS is unable to detect simple patterns that a model-based approach (or even a human briefly looking at the data) would easily infer; this is the cost of having an evaluation technique that places virtually no assumptions on policies.

3.2 Example 2: Systematic Bias Towards Shorter Trajectories

We now show another practically important example where importance sampling can systematically favor policies that assign higher probability to shorter trajectory lengths (in domains where the length of each trajectory may vary). This is a problem that could arise in many practical domains, for example domains where a user is free to leave the system at any time, such as a student doing problems in an educational game or a user chatting with a dialogue system. Moreover, this is especially worrisome when there is some correlation between how long a user stays in the system and the reward that the system obtains. In many cases the reward might be directly proportional to the number of interactions the user has with the system. Even if that is not the case, in many situations, worse policies might bias users to leave the system earlier. For example, in an educational game whose goal is to maximize student learning, we can imagine a policy that gives levels that are too difficult will lead students to leaving the game and hence learning very little, whereas a policy that gives an optimal progression of levels might result in the student to play the game for a longer duration of time and hence learn more. Thus, it is particularly problematic that importance sampling can favor policies that assign higher probability to shorter trajectories even when shorter trajectories are worse than longer ones, which the following example shows to be true.

Consider the domain given in Figure 2. Now suppose we have data collected from a behavior policy π_b that takes each action with probability 0.5. We want to compare two policies: π_X, which takes action a_X with probability 0.99, and π_Y, which takes action a_Y with probability 0.99. Clearly π_Y is the better policy, because it incurs a lot of reward when we encounter trajectories of length 80, while only losing out on a small reward when encountering the short trajectories. Table 1 shows the median estimate, out of 100 simulations, of the Monte Carlo estimator (i.e., the standard on-policy estimator $\hat{V}^{\pi_e}_{\text{MC}} \triangleq \frac{1}{n}\sum_{i=1}^{n}\sum_{t=1}^{T_i} R_{i,t}$), as well as the median IS and WIS estimates using 1000 samples each. We find that while π_Y is, in actuality, much better, IS essentially only weighs the shorter trajectories, so the estimates only reflect how well the policies do on those trajectories. WIS simply (almost) doubles the estimates because half of the samples have extremely low importance weights. So why does this occur? When using IS in settings where trajectories can have varying lengths, the importance weight of shorter trajectories can be much larger than for longer trajectories, because for longer trajectories, we are multiplying more ratios of probabilities that are more often smaller than one. This happens even if the policy we are evaluating is more likely to produce a longer trajectories than a shorter one (because there are exponentially many longer trajectories and so each individual trajectory has an exponentially smaller weight).

4 Guaranteeing Fairness

We will now show conditions under which we can guarantee fairness when using importance sampling for policy selection.

Theorem 4.1. *Using importance sampling for policy selection when we have n samples from the behavior policy is fair provided that*

$$w_{MAX}^{\pi_1} V_{MAX}^{\pi_1} + w_{MAX}^{\pi_2} V_{MAX}^{\pi_2} \leq |V^{\pi_1} - V^{\pi_2}| \sqrt{\frac{2n}{\ln 2}}$$

where w_{MAX}^{π} is the largest importance weight for policy π and V_{MAX}^{π} is the largest possible value for policy π. In other words, Algorithm 1 is fair provided that $\epsilon \leq |V^{\pi_1} - V^{\pi_2}|$ and $\delta \leq 0.5$.

Theorem 4.1 can be shown with a simple application of Hoeffding's inequality. Alternatively, we can use other concentration inequalities to obtain fair algorithms of a similar form. Additionally, we can extend the algorithms to policy selection with more than two policies by applying a union bound, but that is omitted here for brevity. Notice that Theorem 4.1 tells us that as long as neither policy is too far from the behavior policy in terms of the largest possible importance weight, then we can guarantee fairness, which intuitively makes sense; we can only fairly compare policies that are similar to the behavior policy. However, how far we stray will also depend on how different the values of the policies are from each other. This is a quantity we do not know, so we must pick an ϵ where either we think $\epsilon \geq |V^{\pi_1} - V^{\pi_2}|$ or we are comfortable with the possibility of choosing a policy that is worse than ϵ from the better policy. Theorem 4.1 helps us better understand when we can guarantee fairness and gives hope that importance sampling is still useful for policy selection, but there is still much to do before we can implement fair policy selection to obtain decent policies that are very different from the behavior policy, which is what we would often like in practice. Our hope is that our paper will lead to more researchers thinking about the problems that arise in off-policy policy selection and how we may mitigate these problems, which we believe has been largely ignored in the literature.

Algorithm 1 Fair Policy Selection

```
Input: $\pi_1, \pi_2, V_{MAX}^{\pi_1}, V_{MAX}^{\pi_2}, \epsilon, \delta$
       $\tau_1, \tau_2, \ldots, \tau_n \sim \pi_b$
  if $w_{MAX}^{\pi_1} V_{MAX}^{\pi_1} + w_{MAX}^{\pi_2} V_{MAX}^{\pi_2} \leq \epsilon \sqrt{\frac{2n}{\ln 1/\delta}}$ then
    return $\max(\hat{V}_{IS}^{\pi_1}, \hat{V}_{IS}^{\pi_2})$
  else
    return No Fair Comparison
  end if
```

References

[1] Tang Jie and Pieter Abbeel. On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems*, pages 1000–1008, 2010.

[2] Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML (3)*, pages 1–9, 2013.

[3] Travis Mandel, Yun-En Liu, Sergey Levine, Emma Brunskill, and Zoran Popovic. Offline policy evaluation across representations with applications to educational games. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1077–1084. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[4] D. Precup, R. S. Sutton, and S. Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the 17th International Conference on Machine Learning*, pages 759–766, 2000.

[5] P. S. Thomas, G. Theocharous, and M. Ghavamzadeh. High confidence policy improvement. In *International Conference on Machine Learning*, 2015.

[6] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Multi-attribute decisions is best characterized by attribute-wise reinforcement learning model

Shaoming Wang
Department of Psychology
New York University
`shaoming@nyu.edu`

Bob Rehder
Department of Psychology
New York University
`bob.rehder@nyu.edu`

Abstract

Choice alternatives often consist of multiple attributes that vary in their predictiveness of reward. Some standard models assert that decision makers either weigh such attributes optimally (rational models) or use heuristics in which attributes are used suboptimally but in a manner that yields reasonable performance at minimal cost (e.g., the take-the-best heuristic). However, such models have no principled account of how decisions can be overly influenced by recent experiences (e.g., recency effects) or how individuals can end up with different attribute weights. In contrast, standard reinforcement learning models account for recency effects and different attribute weights as a result of different trial presentation orders and patterns of feedback. Yet these models are known to perform poorly with multi-attribute stimuli. Moreover, it remains unclear whether choices are evaluated at the level of attributes or alternatives in multi-attribute framework. We conducted a two-alternative choice experiment with stimuli that varied on three binary attributes. All attributes were predictive of reward but varied in their predictiveness. Participants generally learned to use all three attributes and the relative rank of those attributes. Our analysis also revealed that the time needed to make decisions increased as the number of relevant attributes increased, suggesting that subjects took an attribute-wise approach. Computational model fitting revealed that models that assumed that learners use multiple attributes performed better than those that didn't and that models that account for recency effects performed better than those that didn't. The best performing model was one that incorporated both factors. We discuss the role of selective attention in reinforcement learning more generally and the potential need to incorporate more hypothesis-testing like processes to account for results with multiple-attribute stimuli.

Keywords: multi-attribute decision making, reinforcement learning

1. Introduction

Choice options often consist of multiple attributes that vary on their predictiveness of rewards. One strand of models in decision making—*rational models*—assume perfect learning of how to weight and combine attributes so as to maximize reward [1, 2]. Heuristic models, such as *take-the-best model*, offer suboptimal solutions that consider some attributes while ignore others [3]. Numerous studies have evaluated which of these approaches are used by human decisions makers but have not yielded a clear answer [4]. Moreover, they appear to neglect how decisions can be overly influenced by recent experiences, which has been well-captured by reinforcement learning (RL) models [5]. RL represents an advantage over the models described above in not requiring a full record of past experiences without arbitrarily ignoring information that could facilitate decisions. Yet, despite their successes RL models are notoriously inefficient as the number of attributes in choice options increases—the so-called *curse of dimensionality* [6].

We propose that the curse of dimensionality can be overcome by an *attribute-wise RL model* in which choices are evaluated by evaluating attributes and and then combining those evaluations in an optimal way. Moreover, decisions will be overly influenced by recent experiences, which can be well characterized by RL models. Human participants performed a 2-alternative-forced-choice task with multi-attribute stimuli and probabilistic feedback. We found that participants generally learned to use all attributes and the relative rank of those attributes. We also found that response time increased as the number of discriminating attributes increased, supporting the claim that subjects evaluated choices attribute-wise. Finally, that participants' decisions was best characterized by an RL model implies an effect of recent reward histories. These results were inconsistent with a rational model that assumes participants will use information optimally or a take-the-best model that asserts only one attribute will be used during decision making.

2. Method

2.1 Experimental task

The task stimuli varied on three binary attributes: head (triangular or rectangular); body (light or dark); and tail (big or small). The task consisted of 6 training blocks and a testing block, each 36 trials long. Before the start of training participants were informed that all three attributes (head, body and tail) were predictive of reward and that they needed to learn about the importance of the cues and stimulus attributes through trial-and-error, with the goal to collect as many one-dollar bills as possible. On each training trial participants were presented with two stimuli (aliens), each consisting of one cue on each of the three attributes (Fig. 1 1). The participants' task was to choose the stimulus that they deemed more likely to be rewarded. After a response the reward associated with that choice was displayed, consisting of an image of either a one-dollar bill or a zero-dollar bill. To associate reward with an alien on each trial, we associated with each stimulus dimension a target weight indicating how important that attribute was for predicting reward. Those weights were 1.7776, 1.3332, and 0.8888, for Attribute 1, 2 and 3, respectively. These weights were compensatory, such that Attributes 2 and 3 together outweigh Attribute 1. The assignment of the physical cues and attributes to their logical roles (see below) was counterbalanced across participants. We defined 13 types of choice problems defined by the amount of evidence provided by one alternative (referred to as A) over the other (B). For each choice type, Table 1 defines the evidence that attribute i provides in favor of alternative A, denoted by $Ev(Att_i)$. $Ev(Att_i) = 1$ means that the cues on attribute i provide evidence for A, $Ev(Att_i) = -1$ means that they provide evidence for B, and $Ev(Att_i) = 0$ means they favor neither alternative, a situation that arises when A and B display the same cue on that attribute. Reward probabilities were derived from a logistic regression where the attributes were linearly combined via the weights $\{w_1, w_2, w_3\}$ were linearly combined:

$$P(r|A) = \frac{1}{1 + e^{-\sum_1^3 (w_i * Ev(Att_i))}} \tag{1}$$

$$P(r|B) = 1 - P(r|A) \tag{2}$$

$P(r|A)$ for each of the 13 choice types is presented in Table 1. It also presents the number of times each choice type was presented during the 216 training trials (e.g., choice type 1 was presented 24 times). The number of times each choice type was rewarded was chosen so at to approximate its target (e.g., choice type 1 was rewarded on 21 of its 24 presentations). Finally, Table 1 presents the actual probability of reward for each choice type. The attribute weights recoverable from these 216 training trials were 1.7641, 1.3309 and

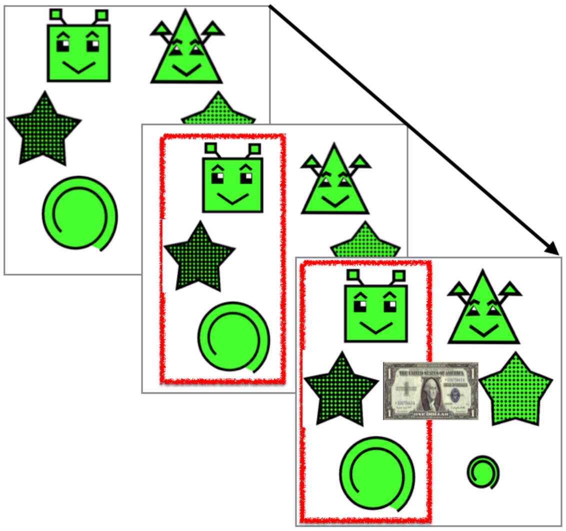


Figure 1: Schematic of a trial.

Choice Type	$Ev(Att_1)$	$Ev(Att_2)$	$Ev(Att_3)$	Target $\Pr(r\|A)$	N	Rewarded	Actual $\Pr(r\|A)$
1	1	0	0	0.855	24	21	0.875
2	0	1	0	0.791	24	19	0.792
3	0	0	1	0.709	24	17	0.708
4	1	1	0	0.957	12	11	0.917
5	1	-1	0	0.609	12	7	0.583
6	1	0	1	0.935	12	11	0.917
7	1	0	-1	0.709	12	9	0.750
8	0	1	1	0.902	12	11	0.917
9	0	1	-1	0.609	12	7	0.583
10	1	1	1	0.982	18	18	1.000
11	1	1	-1	0.902	18	16	0.889
12	1	-1	1	0.791	18	14	0.778
13	-1	1	1	0.609	18	11	0.611

Table 1: Structure of the task.

0.8912. These are the weights that an ideal observer should adopt to maximize reward. Finally, instantiations of the 13 choice types were assigned to blocks of 36 training items such that the proportion of each choice type was the same in each block.

2.2 Computational models

To analyze choice behavior, we explored four computational models, The *Rational model (RAT)* is a variant of a weighted-additive decision model, RAT [2]. It uses all the relevant available information to update cue validity [2] on the basis of the proportion of rewarded inferences made across stimulus pairs in cases where a cue discriminates between alternatives. In our task, the validity of cue i (v_i) was defined as below with a Bayesian modification pointed out by [4]:

$$v_i = \frac{1 + \mathit{rewarded\_decisions}_i}{2 + \mathit{rewarded\_decisions}_i} \tag{3}$$

Validities for all the cues present in a trial were updated using Equation 3, after feedback was given on each trial. If a chosen stimulus was rewarded, the number of rewarded decisions for that cue increased by one, otherwise it remained the same. At the same time, the validities for the unchosen stimulus were also updated, given participants were instructed that only one stimulus would be rewarded on each trial. After calculating cue validities, each cue was assigned a weight (w_i), representing the independent contribution of each cue to the evidence in favor of an alternative. This was given by the log odds of cue validities [2] in Equation 3.

$$w_i = \log\left(\frac{v_i}{1 - v_i}\right) \tag{4}$$

The value of each stimulus was determined by the sum of cue weights for attributes (n is the number of discriminating attributes) that discriminate between alternative in that stimulus:

$$Q(S) = \sum_{i=1}^{n} w_i \tag{5}$$

then these values were entered into a softmax choice function:

$$p(\mathit{choose}\ S) = \frac{e^{\beta * Q(S)}}{\sum_{j=1}^{2} e^{\beta * Q(S_j)}} \tag{6}$$

where the inverse temperature parameter β represents the level of noise in the decision process, with larger β corresponding to low decision noise and near-deterministic choice of the highest-value option, and small β corresponding to high decision noise and nearly random decisions. This model can be thought of an "ideal observer" model as it optimally evaluates the probability of an alternative getting rewarded given cues and associated validities for both alternatives.

The second model is *"take-the-best" model* [2, 3] uses the same rule to update cue validities as rational model. However, instead of using all the cues and attributes, TTB sequentially searches through cues in a descending order of their validities and stops upon finding the highest-ranked cue that discriminates between two alternatives.

The third model is an *alternative-wise reinforcement learning model (atl-RL)* [7] learns values for each distinct stimulus through the standard Rescorla-Wagner learning rule. After choosing a stimulus S and observing the reward for that choice, the value of S, $Q(S)$, was updated according to:

$$Q^{new}(S_{chosen}) = Q^{old}(S_{chosen}) + \alpha\,(R_t - Q^{old}(S_{chosen})) \tag{7}$$

where α is a learning-rate parameter. The unchosen stimulus was updated by the same predition error with reversed sign:

$$Q^{new}(S_{unchosen}) = Q^{old}(S_{unchosen}) - \alpha\,(R_t - Q^{old}(S_{unchosen})) \tag{8}$$

Stimuli values, $Q(S)$, were then entered the same softmax choice rule described above to compute choice probabilities.

The fourth model is an *attribute-wise reinforcement learning model (attr-RL)* [7] differs from the atl-RL model in using the Rescorla-Wagner learning rule to learn the value of individual cues, instead of whole stimuli. The value of stimulus S was calculated as the sum of the values of its three features, $V(f)$, as:

$$Q(S) = \sum_{f \in S} V(f) \tag{9}$$

and a choice is made on the basis of the softmax rule. The value of each feature in a chosen and unchosen stimulus was updated according to *R-W* rule using formula (10) and (11):

$$V^{new}(f) = V^{old}(f) + \alpha\,\left(R_t - V^{old}(f)\right), f\ \in S_{chosen} \tag{10}$$

$$V^{new}(f) = V^{old}(f) - \alpha\,\left(R_t - V^{old}(f)\right), f\ \in S_{unchosen} \tag{11}$$

3. Results

3.1 Choice behavior in testing phase

To assess what participants learned about the three attributes, we computed logistic regression weights from the testing block during which no feedback was provided. Each learner's normalized beta weights are displayed in the simplex plot in Figure 2 [8]. Points reflect the relative contribution of the attributes on choices. The middle cross corresponds to the case where the attributes are equally weighed and the red circles to cases where only one attribute was used. The black asterisk represents optimal performance (weights of 0.443, 0.334, and 0.224). Informal inspection of Figure 2 suggests that in fact most learners made use of all three attributes, contradicting the take-the-best model. Moreover, learners generally recovered the attributes' relative importance. Of the six possible orderings of attribute weights, learners exhibited the optimal ordering (Attribute 1 > 2 > 3; points in blue) more than any other (51% of the participants).

3.2 Response time (RT) in testing phase

We examined response times during the testing block. as a function of choice type. First, to assess the rational and the take-the-best models, we compared RTs within pairs of choice types (4 and 5, 6 and 7, 8 and 9, and 10 and 13). Within each pair, the evidence for choice A is the same if only the "best" attribute is considered (see Table 1); thus the take-the-best model predicts that types within a pair are equally difficult decisions (and so should exhibit equal RTs). In contrast, because it considers all attributes, the rational model predicts that the first member of each pair is less difficult than the second (e.g., choice type 4 is easier than 5) and so should exhibit faster RTs. Figure 3 confirms the predictions of the rational model for each of the four pairs. We also asked whether participants evaluated decisions attribute- or alternative-wise by

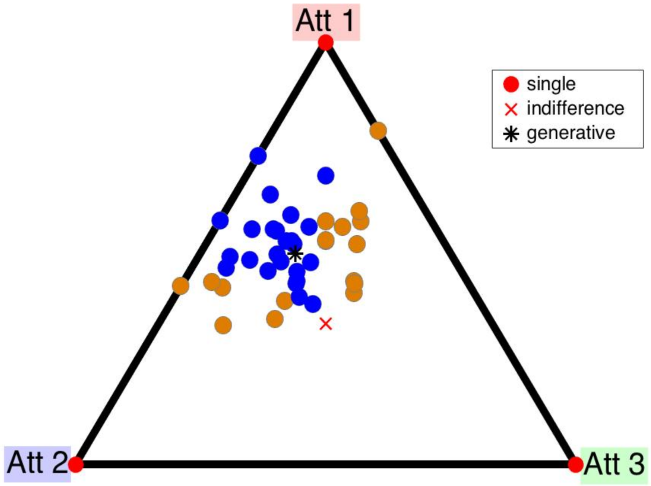


Figure 2: Simplex plot of beta weights.

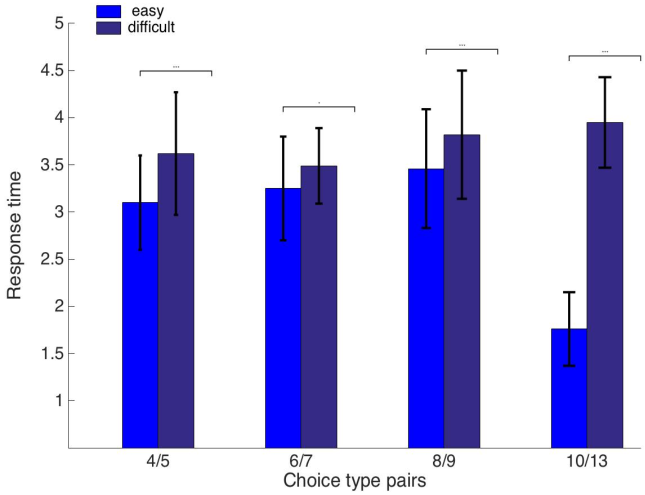


Figure 3: Choice difficulty and RTs.

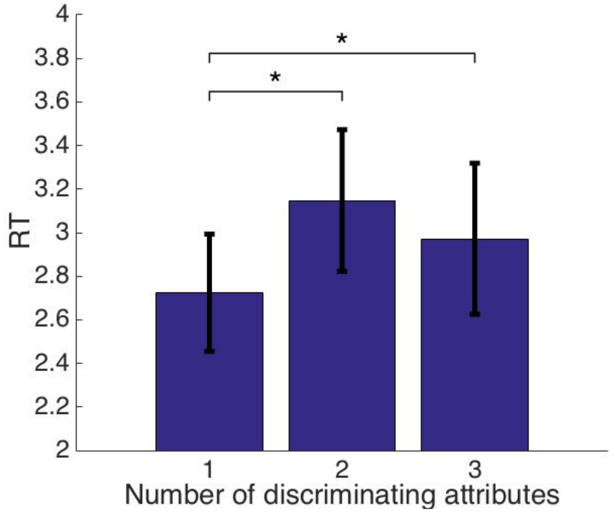
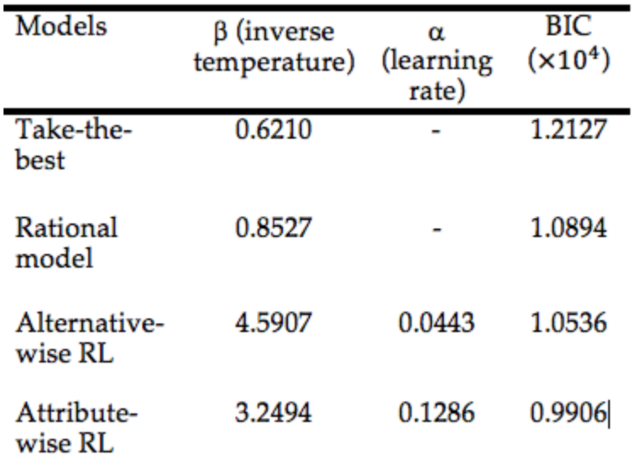


Figure 4: Number of relevant attributes and RTs.



Models	β (inverse temperature)	α (learning rate)	BIC (×10⁴)
Take-the-best	0.6210	-	1.2127
Rational model	0.8527	-	1.0894
Alternative-wise RL	4.5907	0.0443	1.0536
Attribute-wise RL	3.2494	0.1286	0.9906

Table 2: Mean parameter estimation and BIC.

examining choice types that differ in the number of discriminating attributes. If choices were evaluated alternative-wise then RTs should not vary with the numbers of discriminating attributes. In fact, RTs were faster when only one attribute discriminated as compared to two, $t(46) = -2.73$, $p < 0.01$, and three, $t(46) = -3.02$, $p < 0.01$ (Fig. 4); RTs were not significantly different for two and three attributes. These results support the notion that participants learn and use multiple attributes, consistent with RAT and attr-RL but not TTB and alt-RL.

3.3 Model comparison

We fit each participants' choice data during the entire experiment to each of the four models. Model likelihoods were based on assigning choice probabilities on every trial. Table 2 shows the parameter estimation of all the models. We then computed each participant's Bayesian Information Criterion (BIC) as: $BIC = -2\times \ln(L) + K_{param}\times \ln(N)$, where L is the likelihood of the choice probabilities given model and parameters, is the number of parameters in the model, and N is the number of observations (number of trials in training block) for each participant. We then aggregated BIC across the participants to compare between models. The model that yields a smaller BIC is considered to provide a more account of the data. The results (see Table 2) are in support of attribute-wise RL model over other models.

General Discussion

We have proposed that an attribute-wise reinforcement learning model can better characterize human decision making in multi-attribute, probabilistic environment. We have shown that participants could learn the task and correctly rank the order of attributes in terms of their objective importance. Analyses revealed that the difficulty of decisions, as well as the number of relevant attributes, influenced the time participants needed to make decisions. These results suggest that rather than merely "taking the best" [2], our subjects made use of multiple sources of available information to make decisions. They also support the idea that participants made decisions based on attributes instead of alternatives (i.e., whole stimuli). Our model fits revealed that an attribute-wise RL model provided the best fits to data from both the training and testing phases. This model represents individual attributes rather than alternatives and sums the values of attributes in order to compute the value of alternatives. Moreover, the better fit of RL models as compared to the rational models suggests that recent experiences had greater influence on decisions, as already documented in the decision-making literature. Taken together, these results suggest that humans can adopt an attribute-wise, computational efficient process to guide decisions in probabilistic, multi-attribute environment.

Further analysis should be conducted to address individual differences that are ubiquitous in studies of human choice behavior. Previous studies [4] have found dynamically strategic shifting during learning and decision making. Consistent with these findings, preliminary trial-by-trial analysis in our study (not presented here) revealed that the between-subject variability was greater than predicted by either TTB or RAT. Given the effects of recency they incorporate, the end-state predictions of RL models will exhibit greater sensitivity to effects of trial order and thus between-subject variability. Yet, a full account of such variability may require incorporating additional cognitive mechanisms, such as the operation of selective attention that has played a central role in category learning and classification, or even hypothesis testing-like mechanisms in which subjects explicitly abandon use of one subset of attributes in favor one another [9, 10]. Indeed, our trial-by-trial analysis revealed that learners sometimes exhibited shifts in the ranking of their subjective attribute weights. Further computational modeling analysis should be conducted to investigate these possibilities.

References

[1] Simon, H. A. (1956). Rational choice and the structure of the environment. Psychological Review, 63, 129 –138.

[2] Lee, M. D., & Cummins, T. D. R. (2004). Evidence accumulation in decision making: Unifying the "take the best" and the "rational" models. Psychonomic Bulletin & Review, 11, 343–352.

[3] Gigerenzer, G., & Goldstein, D. G. (1996). Reasoning the fast and frugal way: Models of bounded rationality. Psychological Review, 103, 650–669.

[4] Bergert, F. B., & Nosofsky, R. M. (2007). A response-time approach to comparing generalized rational and take-the-best models of decision making. Journal of Experimental Psychology: Learning, Memory, and Cognition, 33, 107–129.

[5] Daw ND, Niv Y, Dayan P. 2005. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. Nature Neuroscience 8:1704–11

[6] Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. Cambridge, MA: MIT.

[7] Niv, Y., Daniel, R., Geana, A., Gershman, S.J., Leong, Y.C., Radulescu, A., and Wilson, R.C. (2015). Reinforcement learning in multidimensional environments relies on attention mechanisms. Journal of Neuroscience. 35, 8145–8157.

[8] Coenen, A., Rehder, B., Gureckis, T. (2015). Strategies to intervene on causal systems are adaptively selected. Cognitive Psycholog, 79, 102-133.

[9] Kruschke, J. K., & Johansen, M. K. (1999). A model of probabilistic category learning. Journal of Experimental Psychology: Learning, Memory and Cognition, 25(5), 1083–1119.

[10] Rehder, B. & Hoffman, A.B. (2005). Eyetracking and selective attention in category learning. *Cognitive Psychology*, *51*, 1-41.

State Space Decomposition and Subgoal Creation for Transfer in Deep Reinforcement Learning

Himanshu Sahni[*]
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
`himanshu@gatech.edu`

Saurabh Kumar*
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
`skumar311@gatech.edu`

Farhan Tejani
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
`farhantejani@gatech.edu`

Yannick Schroecker
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
`yannickschroecker@gatech.edu`

Charles Isbell
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
`isbell@cc.gatech.edu`

Abstract

Typical reinforcement learning (RL) agents learn to complete tasks specified by reward functions tailored to their domain. As such, the policies they learn do not generalize even to similar domains. To address this issue, we develop a framework through which a deep RL agent learns to generalize policies from smaller, simpler domains to more complex ones using a recurrent attention mechanism. The task is presented to the agent as an image and an instruction specifying the goal. This meta-controller guides the agent towards its goal by designing a sequence of smaller subtasks on the part of the state space within the attention, effectively decomposing it. As a baseline, we consider a setup without attention as well. Our experiments show that the meta-controller learns to create subgoals within the attention.

Keywords: Hierarchical Reinforcement Learning, Transfer Learning, Policy Gradient, Attention Mechanism

---

[*] Denotes equal contribution.

1 Introduction

Usually, reinforcement learning (RL) agents cannot generalize policies learned in small domains to larger, more complicated ones. Yet this ability can allow them to immediately apply skills learned in simple settings without having to explore a large state space. In the deep learning setting, reduction in the size of the state also means smaller networks are required, which are easier to train. In this work, we present an approach to decompose complicated environments into simpler ones and provide subgoals within them that ultimately solve the larger task. The agent can be pre-trained on the smaller environment to solve each subgoal independently, or in conjunction with the subgoal creation algorithm.

We describe a meta-controller that learns to decompose the state space and provide subgoals solvable within the smaller space. The meta-controller is solving a delayed reward problem as it only gets positive reinforcement when the underlying agent solves the original task. It has to come up with a sequence of subgoals which maximizes the expectation of this reinforcement. In addition to creating subgoals, the meta-controller also fragments the state space such that the underlying agent is presented with a smaller state on which it can easily learn an optimal policy for the subgoal. It does this by using an attention mechanism, similar to the Recurrent Attention Model [1]. The meta-controller learns to control its attention and only passes the part of the state within it to the agent. The meta-controller's MDP formulation is:

- States, S, are summaries of its past and current attentions.
- Actions, A, are the locations of the attention L^{attn}, and a distribution over the set of subgoals, **g**.
- Rewards, r, are positive if the underlying agent solves the task and a small negative step cost otherwise
- Transitions: The underlying agent executes its policy according to the state and subgoal provided to it. Since this policy is unknown to the meta-controller this is a source of stochasticity in its environment.

The meta-controller selects a value for L^{attn} and a distribution $P(g)$. The state space under that location and a subgoal, g, are passed to the underlying agent. The agent then chooses an atomic action that moves it towards achieving g. The new agent location L^{agent} changes the meta-controller's environment, which picks a new attention and subgoal.

In this work, we make a few simplifying assumptions. First, we assume that the underlying agent has access to the optimal policy for each subgoal. Such a goal-dependent policy can be learned by a technique such as universal value function approximators (UVFAs) [2]. UVFAs learn to approximate $V(S, g)$, or the value function with respect to a goal, using a function approximator such as deep neural nets. The learned value function $V(S, g; \theta)$ can be used to construct a policy that achieves the goal g. This value function can be trained independent of or in conjunction with the meta-controller by providing intrinsic rewards for achieving subgoals [3]. Secondly, we assume that the agent remains still unless both its location and the subgoal are present within the state provided to it by the meta-controller. In general, the meta-controller will automatically be incentivized to focus its attention and provide subgoals such that the underlying agent is able to solve the given task because of its reward structure. In this case, that means keeping both the agent location and the subgoal within the attention. For example, in the game of Pacman, if the subgoal is to eat the closest pill, the underlying agent should have both the Pacman and at least one pill within the state provided to it. Otherwise, the agent may move randomly, and it will be unable to achieve its overall goal of getting a high score.

The above assumptions simplify training of the meta-controller, but the methodology we provide should be applicable in the general setting where the policy of the underlying agent is learned as well.

2 Related Work

Our work most closely matches that of Kulkarni et al. [3]. They present a hierarchical framework where an agent learns from intrinsic rewards provided by a higher level agent setting subgoals and operating on a longer time frame. Rewards for the higher level agent are provided by the environment for completing tasks. The subgoals in turn are provided through functions over entities and relations in an object oriented framework. In a sense, our approach takes this a step further by decomposing the state space such that the base agent has to only see a small portion of it at any time. This allows for better computational efficiency, as the base agent can now use smaller networks, and may allow for transfer of learnt policies to different parts of the state space that are similar without having to explicitly explore them. To achieve this, the higher level agent, or meta-controller, must learn to integrate information over the states it has observed so far. Therefore, we use a recurrent model to represent the meta-controller through a Long Short Term Memory (LSTM) network [4]. Kulkarni et al. use a pair of DQNs for the agent and meta-controller.

In order to train the attention mechanism of our meta-controller, we employ a technique similar to that of Mnih et al. [1]. They use policy gradients to train an attention mechanism for classification and simple control tasks. In our approach, we do not employ a complex glimpse sensor, but instead simply use a 5x5 crop of our input image. This can be incorporated into our setup easily. Further, instead of specifying L^{attn} directly using a continuous output, we use discrete actions, $up, down, noop$, to move the attention.

Finally, Schaul et al. [2] describe how goal specific function approximators may be constructed for deep RL agents. Such a function can be constructed for our base learner by independently learning subgoals on a 5x5 image. We do not provide results for this setting in this paper but it can be integrated into future work.

3 Preliminaries

Before proceeding further, we provide a brief introduction to reinforcement learning and policy gradients.

3.1 Reinforcement Learning and MDPs

Reinforcement learning addresses the problem of choosing behavior that maximizes some notion of long term cumulative reward. It is typically formulated as a Markov decision process (MDP). An MDP is characterized by the tuple $< S, A, T, R, \gamma >$. S is the set of states an agent can be in. $A(S)$ is the set of actions the agent has available to it in each state. Typically, the agent chooses an action to execute from A, which may lead it into a new state according to the transition function $T : S \times A \to P(S)$. $R : S \times A \to \mathbb{R}$ is a scalar value received upon executing an action in a state. Finally, $0 \leq \gamma \leq 1$ is the discount factor. A policy, $\pi : S \to P(A)$, informs the agent on which action to execute in each state. The goal of a reinforcement learning agent is to find π^*, the optimal policy, which maximizes the long term expected reward, or utility, in each state.

3.2 Policy Gradients

Policy gradient methods look to directly directly maximize expected reward by adjusting policy parameters. The expected reward for a trajectory sampled from a policy π parametrized by θ is given by

$$J(\theta) = \mathbb{E}_{p(S_{1:T};\theta)} \sum_{t=1}^{T} r_t = \mathbb{E}_{p(S_{1:T};\theta)}[R] \tag{1}$$

where, $p(S_{1:T};\theta)$ depends on the state distribution induced by π, and R is the return for the trajectory. In this formulation, no discounting is performed. The policy gradient theorem [5] states that the gradient for J is given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{p(S_{1:T};\theta)} \sum_{t=1}^{T} \nabla_\theta log[\pi(a_t|s_{1:t};\theta)]R_t \tag{2}$$

The expectation can be approximated by sampling a batch of trajectories from the current policy and averaging the gradients over them. This is the REINFORCE algorithm [6]. The vanilla policy gradient may suffer from high variance as it relies on Monte Carlo samples. A common modification to reduce variance is to subtract a baseline from the returns. The baseline b_t is computed by taking the average of observed returns over the past N time steps: $\frac{1}{N}\sum_{n=T-N+1}^{T} r_{t_n}$. In this work, we chose $N = 100$.

4 Environment

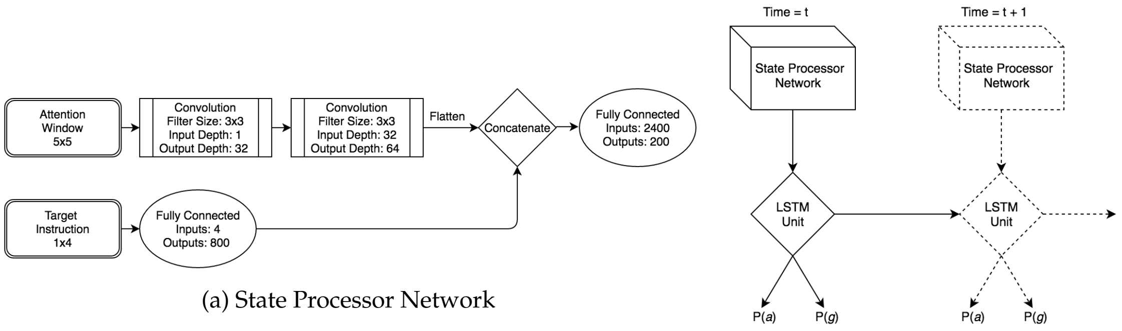
For our experiments, we use an environment consisting of a 10x5 grid. The grid consists of four ”rooms,” where each room is a horizontal kx5 strip for some k not exceeding 4. The rooms are stacked on top of each other and are each a different color from the set $\{red, green, blue, yellow\}$. The environment also generates an instruction as a one-hot vector of length 4, specifying the target room. An episode terminates when either the agent reaches the target room, receiving a positive reward of $+1$, or it times out without reaching it. The step cost is -0.01.

5 Approach

We construct three frameworks for the meta-controller that is tasked with providing subgoals to the underlying agent such that it navigates successfully to the target room. In all experiments, the meta-controller uses the Adam Optimizer with a learning rate of $1 \times e^{-5}$. The agent and attention (if used) always start of at the top left corner of the grid.

5.1 Meta-Controller with No Attention Mechanism

First, we simplify the problem by providing the entire state space as input to the meta-controller at each time step, therefore not employing any attention mechanism. Specifically, the meta-controller receives as input the 10x5 image of the grid which contains the rooms as well as the location of the agent, L^{agent}. The output of the meta-controller is a distribution $P(g)$ over the rooms. A room is sampled from $P(g)$ and is provided to the base agent as an instruction. This is the subgoal it must achieve. It is assumed that the underlying agent can move optimally on the entire 10x5 grid given an instruction. In this setup, the optimal policy for the meta-controller is to always output the target room directly.


Attention Window 5x5
Convolution Filter Size: 3x3 Input Depth: 1 Output Depth: 32
Convolution Filter Size: 3x3 Input Depth: 32 Output Depth: 64
Flatten
Concatenate
Fully Connected Inputs: 2400 Outputs: 200
Target Instruction 1x4
Fully Connected Inputs: 4 Outputs: 800
Time = t
Time = t + 1
State Processor Network
State Processor Network
LSTM Unit
LSTM Unit
P(a)
P(g)
P(a)
P(g)

(a) State Processor Network

(b) Architecture of Meta-Controller with Attention

Figure 1: Network architecture for the partial state decomposition and constrained attention mechanism experiments. The $5x5$ attention window and target instruction are inputted at each time step into the meta-controller, which then outputs probability distributions over attention actions and subgoal instructions.

5.2 Meta-Controller with Partial State Decomposition

In this setup, we have a meta-controller with an attention mechanism, which consists of a 5x5 window into the 10x5 grid. In addition to the subgoal instruction, it must now also output an action to control the attention. Here, the attention mechanism only partially decomposes the state space, meaning that the agent may move optimally to a provided subgoal, even if the agent is not within the current attention. The subgoal, however, must be located inside the attention. The goal of the meta-controller is to to find the location of the target room using its attention mechanism and then instruct the agent to go to that room color at every time step.

The architecture of this meta-controller consists of a state processor network, which takes the $5x5$ attention window and target instruction provided by the environment as input at each time step. It processes these inputs using a feedforward convolutional network and uses an LSTM unit to output $P(g)$ and $P(a)$, where $P(a)$ is the probability distribution over attention actions. The convolutional layers of the network use rectified linear unit activation functions. The attention action affects the next attention location, L^{attn}, while the subgoal instruction affects the next agent location, L^{agent}. Thus, the LSTM's hidden state contains knowledge gained from taking a sequence of both instruction and attention actions in an episode. In order to train this network effectively with Policy Gradients, we assume that the attention and instruction actions probabilities are independent of each other.

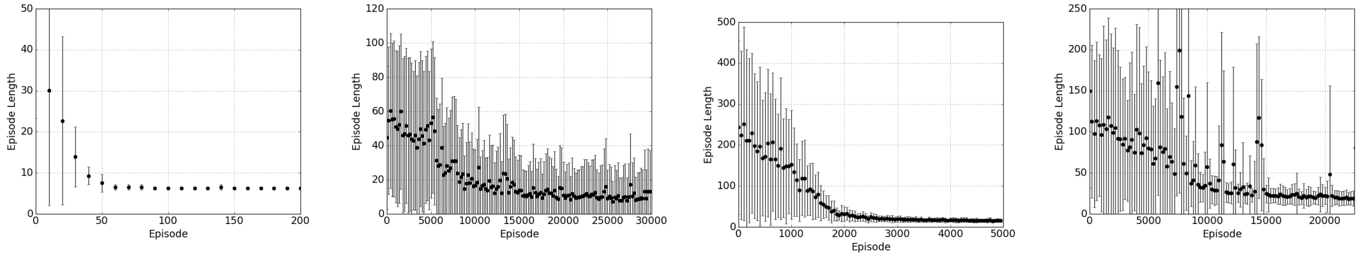
5.3 Meta-Controller with Constrained Attention Mechanism

In this setup the agent will not move unless it is within the attention. This means that the meta-controller must instruct the agent to move to a room within the view of the attention before moving downwards and repeating the process until the agent has reached the target room. Thus, if the agent or the subgoal do not appear within the decomposed state space, the target task will not be achieved. In both the partial state decomposition and constrained attention mechanism experiments, the LSTM unit allows the meta-controller to use its memory of the locations of the agent and target room to guide its action selection when either the agent or target room is not present within the attention window at a particular time step. The constrained attention mechnism framework adds the additional step of constructing an optimal sequence of subgoals for the agent to reach the target room, and this is the overall goal of this paper.

6 Results

6.1 Meta-Controller with No Attention Mechanism

We ran two experiments using the meta-controller with no attention mechanism. In the first experiment, the environment is fixed, i.e. the room arrangement is fixed between episodes, and the target room is always at the very bottom. The optimal policy of the meta-controller is simply to output the instruction corresponding to the target room, since the underlying agent is an optimal agent on the entire grid. In the second experiment, the environment is dynamic, which means that the room arrangement is randomly generated between episodes, but the target room is always located at the very bottom. Here, it must learn a mapping between the color of the bottom-most room and the optimal instruction. These experiments serve as a baseline for the experiments that we run using the meta-controller with the attention mechanism. For both cases, the meta-controller converges to the optimal policy, guiding the agent to the correct room.



(a) Meta-Controller with No Attention Mechanism on a Fixed Environment

(b) Meta-Controller with No Attention Mechanism on a Dynamic Environment

(c) Meta-Controller with Partial State Decomposition

(d) Meta-Controller with Constrained Attention Mechanism

Figure 2: Episode lengths over training episodes. The dots represent means over a variable number of episodes dependent on the total number of episodes displayed and the whiskers represent the variance. In the case of the meta-controller with no attention (a and b), it converges quickly on a fixed environment, supplying the agent with the target room and hence completing episodes in the minimum time. It takes longer to converge in the dynamic case. The meta-controller with attention plots (c and d) show the results on the effect of using attention to guide subgoal creation. For these experiments, the environment was kept fixed.

6.2 Meta-Controller with Attention Mechanism

Here, we show the results for the meta-controller with both the partial state decomposition and the constrained attention mechanism. In these experiments, we keep the environment fixed between episodes. Compared to Figure 2a, in both cases, it takes longer to train the meta-controller to output subgoals leading to the optimal policy. One reason for this is that the meta-controller now has to control its attention in addition to creating subgoals. It is also operating in a partially observed setting and has to integrate information gleaned in past attentions into its hidden state. Note that the state space of the meta-controller is the set of combinations of attention window, target instruction, and hidden state of the LSTM unit. But since this is possible to learn, the underlying agent does not have to be trained on the entire $10\text{x}5$ image, but only on the $5\text{x}5$ attention size. Our approach may scale to even larger sized domains, where directly learning a policy on the original input image may be infeasible.

7 Conclusion

Our overall contribution is a framework that allows an agent to complete a task in a large environment given knowledge of how to do so in a smaller environment. Through the use of an attention mechanism, smaller networks are required, which are easier to train. With all three frameworks developed, the meta-controller learns the representation of the room colors and how that representation transfers to sub-instructions that lead the agent to the desired goal. Our results show that it is possible to scale a policy learned on a smaller environment by decomposing a large state space using an attention mechanism. Our eventual goal is to train the underlying agent in conjunction with the meta-controller and apply this framework to dynamic and complex environments.

References

[1] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. *CoRR*, abs/1406.6247, 2014.

[2] Tom Schaul, Dan Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1312–1320. JMLR.org, 2015.

[3] Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *CoRR*, abs/1604.06057, 2016.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[5] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[6] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

A reward shaping method for promoting metacognitive learning

Falk Lieder[1]
Dept. of Psychology, UC Berkeley
`falk.lieder@berkeley.edu`

Paul M. Krueger[1]
Dept. of Psychology, UC Berkeley
`pmk@berkeley.edu`

Frederick Callaway[1]
Dept. of Psychology, UC Berkeley
`fredcallaway@berkeley.edu`

Thomas L. Griffiths
Dept. of Psychology, UC Berkeley
`tom_griffiths@berkeley.edu`

[1] **These authors contributed equally.**

Abstract

The human mind has an impressive ability to improve itself based on experience, but this potential for cognitive growth is rarely fully realized. Cognitive training programs seek to tap into this unrealized potential but their theoretical foundation is incomplete and the scientific findings on their effectiveness are mixed. Recent work suggests that mechanisms by which people learn to think and decide better can be understood in terms of metacognitive reinforcement learning. This perspective allow us to translate the theory of reward shaping developed in machine learning into a computational method for designing feedback structures for effective cognitive training. Concretely, our method applies the shaping theorem for accelerating model-free reinforcement learning to a meta-decision problem whose actions are computations that update the decision-maker's probabilistic beliefs about the returns of alternative courses of action. As a proof of concept, we show that our method can be applied to accelerate learning to plan in an environment similar to a grid world where every location contained a reward. To measure and give feedback on people's planning process, each reward was initially occluded and had to be revealed by clicking on the corresponding location. We found that participants in the feedback condition learned faster to deliberate more and consequently reaped higher rewards and identified the optimal sequence of moves more frequently. These findings inspire optimism that meta-level reward shaping might provide a principled theoretical foundation for cognitive training and enable more effective interventions for improving the human mind by giving feedback that is optimized for promoting metacognitive reinforcement learning.

Keywords: metacognitive reinforcement learning; meta-decision making; cognitive training; reward shaping; feedback

Acknowledgements

This work was supported by grant number ONR MURI N00014-13-1-0341 and a grant from the Templeton World Charity Foundation. The authors thank Thomas Hills, Peter Dayan, Rika Antonova, Silvia Bunge, Stuart Russell, Smitha Milli, Amitai Shenhav, and Sebastian Musslick for feedback and discussions.

1 Introduction and Background

One of the most remarkable aspects of the human mind is its ability to improve itself based on experience. Such learning occurs in a range of domains, from simple stimulus-response mappings, motor skills, and perceptual abilities, to problem solving, cognitive control, and learning itself (Green & Bavelier, 2008; Bavelier, Green, Pouget, & Schrater, 2012). Demonstrations of cognitive and brain plasticity have inspired cognitive training programs. The success of cognitive training has been mixed and the underlying learning mechanisms are not well understood (Owen et al., 2010; Anguera et al., 2013; Morrison & Chein, 2011). Feedback is an important component of many effective cognitive training programs, but it remains unclear what makes some feedback structures more effective than others, and there is no principled method for designing optimal feedback structures.

Recent work suggests that cognitive growth can be fruitfully conceptualized in terms of a metacognitive reinforcement learning mechanism that identifies which cognitive operation should be executed depending on the current mental state (Krueger, Lieder, & Griffiths, 2017). One instance of metacognitive reinforcement learning is learning how to decide. This includes learning when to rely on habits versus planning and learning how far to plan ahead. Concretely, Krueger et al. (2017) found that an approximate Q-learning algorithm qualitatively captured how quickly people learn to plan farther ahead depending on the reward structure of their training problems. This perspective allows us to translate methods for accelerating reinforcement learning in robots (Ng, Harada, & Russell, 1999) into feedback structures for cognitive training in humans.

Reinforcement learning problems are commonly modeled as Markov decision processes (MDPs) which are defined by a set of states S, a set of actions A, transition probabilities T, a reward function r, and a discount factor γ (Sutton & Barto, 1998). The goal is to learn a policy $\pi : S \mapsto A$ that attains the highest possible sum of (discounted) rewards $V^{\star}(s)$ that can be attained starting from any state s. Ng et al. (1999) showed the model-free reinforcement learning can be accelerated without changing the optimal policy by adding pseudo-rewards (PR) of the form

$$\mathrm{PR}(s, a, s') = \gamma \cdot \hat{V}(s') - \hat{V}(s) \tag{1}$$

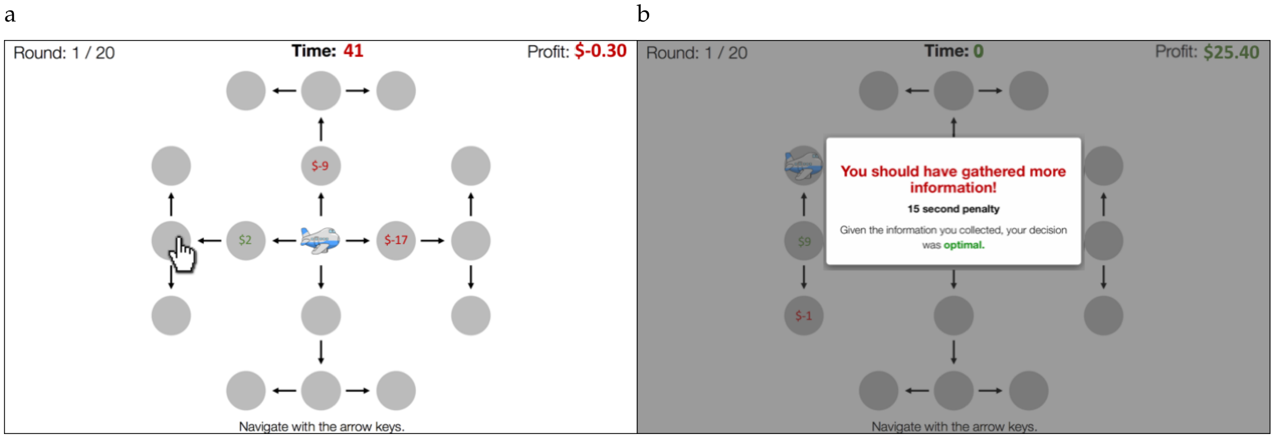
to the reward function r where $\hat{V}$ is an approximation to $V^{\star}$. For instance, a robot learning to navigate a maze may receive a positive PR for moving towards its target location and a negative PR for moving away from it. The resulting reward structure $r'(s, a, s) = r(s, a, s') + \mathrm{PR}(s, a, s')$ makes the optimal policy easier to learn because it aligns immediate reward with long-term value.

Previous work suggested that these pseudo-rewards can also be used to improve human decision-making (Lieder & Griffiths, 2016). Concretely, Lieder and Griffiths (2016) found that presenting pseudo-rewards as incentives increases decision-quality. While this intervention aligned people's choices with the optimal policy, it is unlikely to have improved their planning process in ways that would benefit them in novel environments, because the incentives eliminated the need for planning. Hence, to promote learning to plan it might be more effective to let people plan how to solve the original problem and provide feedback on their planning process. Providing such feedback is challenging for two reasons: first, planning processes cannot be observed directly, and second planning performance is difficult to score automatically. To overcome this first challenge, we leverage the newly developed Mouselab-MDP paradigm (Callaway, Lieder, Krueger, & Griffiths, 2017) that makes it possible to measure people's planning processes by tracing the information they acquire during planning (see Figure 1). To overcome the second challenge, we model optimal planning as the solution to a meta-level MDP (Hay, Russell, Tolpin, & Shimony, 2012) whose actions are cognitive operations and whose states are the planner's beliefs about the rewards it can harvest from the environment by taking a certain action in a certain state. A meta-level MDP M formalizes the problem of selecting the optimal sequences of computations for a resource-bounded computational architecture. This is known as rational metareasoning in the artificial intelligence literature (Russell & Wefald, 1991). Concretely, a meta-level MDP

$$M_{\text{meta}} = (\mathcal{B}, \mathcal{C}, T_{\text{meta}}, r_{\text{meta}}) \tag{2}$$

is a Markov decision process whose actions $\mathcal{C}$ are cognitive operations, states $\mathcal{B}$ represent the agent's probabilistic beliefs, and transition function T_{meta} models how cognitive operations change the agent's beliefs. In addition to a set of computations C that update the agent's belief, the cognitive operations also include the meta-level action $\perp$ that terminates deliberation and translates the current belief into action. The meta-level state b_t encodes the agent's probabilistic beliefs about the domain it is reasoning about. The meta-level reward function r_{meta} captures the cost of thinking and the reward r the agent expects to receive from the environment when it stops deliberating and takes action. The computations C do not yield any external reward. Their only effect is to update the agent's beliefs. Hence, the meta-level reward for performing a computation $c \in C$ is $r_{\text{meta}}(b_t, c) = -\text{cost}(c)$. By contrast, terminating deliberation and taking action ($\perp$) does not update the agent's belief. Instead, its value lies in the anticipated reward for taking action, that is $r_{\text{meta}}(b_t, \perp) = \arg\max_a b_t^{(\mu)}(a)$, where $b_t^{(\mu)}(a)$ is the expected reward of taking action a according to the belief b_t.

Here, we apply reward-shaping to a meta-level MDP of the planning process to derive an optimal feedback structure for accelerating metacogntive reinforcement learning in people.



You should have gathered more information!

15 second penalty

Given the information you collected, your decision was optimal.

Figure 1: Illustration of Mouselab-MDP paradigm used to trace and give feedback on how people plan. Participants plan a sequence of three moves so as to maximize the sum of rewards along the path while minimizing the cost of planning. Each trial starts with a new set of occluded rewards. To observe these rewards participants have to click on them (a) and are provided with feedback after each flight (b). Each click costs 10 cents.

2 A Method for Designing Optimal Feedback Structures for Metacognitive RL

Building on the research summarized above, our method for designing optimal feedback structures to promote metacognitive learning proceeds in the following four steps:

1. Model the cognitive function to be improved (e.g., planning) and the available cognitive operations (e.g., determining the reward for taking a certain action in a certain state) and their costs as a meta-level MDP.
2. Approximate the optimal value function $V^\star_{\text{meta}}$ of the meta-level MDP.
3. Translate the approximate meta-level value function $\hat{V}_{\text{meta}}$ into integrated meta-level pseudo-rewards
$$\text{PR}(b_t, c_t) = \mathbb{E}\left[\hat{V}_{\text{meta}}(B_{t+1})|b_t\right] - \hat{V}_{\text{meta}}(b_t) + \mathbb{E}\left[r_{\text{meta}}(b_t, c, b_{t+1})\right]$$
4. Translate these PRs into rewards and punishments that provide meaningful feedback to people (e.g., Figure 1b).

This method is very general and widely applicable. Here we apply it to accelerating the process by which people learn to plan in the Mouselab-MDP paradigm (Callaway et al., 2017). In this paradigm the task is to find a sequence of moves (m) to collect as much reward as possible from the locations (l) along the resulting path (see Figure 1a). Critically, all of the rewards r are initially occluded. They can be uncovered by clicking on their location and each click has a cost.

First, we model deciding how to plan under cognitive constraints as a meta-level MDP. The set of cognitive operations $\mathcal{C}$ comprises the computations $C = \{c_1, \cdots, c_n\}$, each of which retrieves the reward of one of the n locations (e.g., by clicking on one of the locations in Figure 1a) and updates the belief state accordingly, and the operation $\perp$ which executes the move with the highest expected return according to the current belief state b_t. The belief state $b_t \in \mathcal{B}$ is defined by the means $\mu_{l,m}$ and variances $\sigma_{l,m}$ of normal distributions on the Q-values $Q(l, m)$ of performing move m in location l of the environment shown in Figure 1 and the current location l. The probability of the next belief state b_{t+1} given the current belief state b_t and a computation c_t, that is $T_{\text{meta}}(b_t, c_t, b_{t+1})$, is defined by the assumptions that the reward of the inspected location will be sampled from the reward distribution $P(R) = \mathcal{N}(\mu_R, \sigma_R)$ and the belief state will be updated according the the laws of probability theory. The meta-level reward $r_{\text{meta}}(b_t, c_t)$ of performing a computation was set to a constant $-\lambda$ which represents the cost of generating and processing another piece of of information. The meta-level reward for acting is the expected reward of the resulting action minus the cost of planning, that is $r_{\text{meta}}(b_t, \perp) = \mathbb{E}\left[r(l, \arg\max_m \mu_{l,m}^{(b_t)})\right] - \tau \cdot \sum_{\mathbf{m} \in M_l} \#\mathbf{m}$,where τ is the planning cost per step, M_l is the set of paths starting from location l, and $\#\mathbf{m}$ is the length of the path $\mathbf{m}$. Here, we set the parameters of this meta-level MDP such that the benefit of complete planning according to perfect information far outweigh its cost (i.e., $\mu_R = 4.50, \sigma_R = 10.60, \lambda = 0.10$, and $\tau = 0.01$). Because of the chosen parameters we can approximate the optimal meta-level value function under the assumption that the agent will deliberate enough to eventually select the best move. We thus approximate the value of performing computation c_t in belief state b_t by

$$\hat{V}_{\text{meta}}(b_t) = \mathbb{E}\left[\max_m Q(l_t, m)|b_t\right] - \lambda \cdot \#U_{b_t} - \tau \cdot \sum_{\mathbf{m} \in M_l} \#\mathbf{m}, \tag{3}$$

where $\#U_{b_t}$ is the number of unobserved locations ahead of the current location.

The resulting pseudo-rewards $\mathrm{PR}(b_t, c_t, b_{t+1}) = \mathbb{E}\left[\hat{V}_{\text{meta}}(B_{t+1})|b_t\right] - \hat{V}_{\text{meta}}(v_t) + \mathbb{E}\left[r_{\text{meta}}(b_t, c, b_{t+1})\right]$ satisfy intuitive desiderata: They are negative when the agent decides prematurely, wastes time on irrelevant information, or takes an action that is suboptimal with respect to the current belief state. Since the rewards of the task environment (Figure 1) are monetary, the meta-level pseudo-rewards can be interpreted as the expected future monetary consequences of the cognitive operations the participant performed or failed to perform. We can therefore convert the pseudo-rewards into a delay-penalty by assuming that our participants value each hour of their time at about \$10. Critically, pseudo-rewards for thinking reflect the expected gain in decision quality regardless of whether it made the person more optimistic or more pessimistic. Next, we evaluate whether the resulting feedback can accelerate the rate at which people learn to plan better in the Mouselab-MDP paradigm shown in Figure 1.

3 Experimental Test of the Method's Efficacy

3.1 Methods

We recruited 60 participants on Amazon Mechanical Turk. The task took 29.7 minutes on average and participants were paid \$1.50 plus a performance-dependent bonus equal to 5% of their earning from one randomly selected trial; the average bonus was \$1.76. There were two experimental conditions: In the feedback condition, participants received a delay penalty computed by the reward shaping method described above along with feedback messages. In the control condition, the delay was constant regardless of performance and participants received no feedback. The assignment of participants to conditions was counterbalanced using psiTurk (Gureckis et al., 2016). All participants solved twenty planning problems that were rendered using the Mouselab-MDP plug-in for JsPsych (Callaway et al., 2017). In each problem, participants had to route an airplane from the center of the screen to one of eight final destinations, with two additional stops along the way (Figure 1a). Each state adds or subtracts money, and the value of each state is initially occluded, but can be revealed at any time by clicking on it. Each click incurs a cost of $\$0.10$. Participants were required to spend at least 45 seconds on every trial, to ensure that clicking doesn't incur a time cost.

The twenty planning problems differed only in the reward function. These reward functions were generated independently of one another and their order was randomized across participants. All rewards were integers between -20 and $+18$ with mean 4.2 and standard deviation 10.3. The reward functions were designed such that three-step planning based on complete information led to significantly higher average returns than 2-step planning, 1-step planning, and random choice ($\$38.35$ vs. $\$16.06$–$\$18.30$). One-step planning performed at chance level.

In the feedback condition, we calculated a pseudo-reward for each of the participant's clicks and moves according to meta-level reward shaping. Whenever the participant chose a move the sum of the pseudo-rewards obtained since the last move was translated into a delay penalty according to a conversion rate of $\$10$ per hour. If the resulting penalty was less than 1 second participants could move on immediately. Otherwise, the move was followed by a timeout period that lasted for the computed duration. During this period the participant could not proceed with the task, the countdown on the minimum trial duration stopped, and a feedback message was shown. As illustrated in Figure 1b, this message gave feedback on whether the participant had considered too few possible paths, whether they had considered irrelevant states, and whether their decision was suboptimal relative to what they knew at the time.

3.2 Results

We quantified how much planning our participants performed by how many of the 16 relevant locations they inspected (# clicks). The quality of the resulting decisions was measured by the relative reward $\frac{r-r_{\text{min}}}{r_{\text{max}}-r_{\text{min}}}$ (where r_{max} and r_{min} are the highest and the lowest total return achievable on that trial) and by the frequency with which participants selected the optimal sequences of moves (% optimal routes).

We found that participants in both conditions learned to collect more information and make better plans (Figure 2). To quantify these effects we fitted the learning curves $y(t)$ shown in Figure 2 by a sigmoidal function of trial number t,

$$\hat{y}(t) = (1-\beta_0) \cdot \frac{1}{1+\exp\left(-(\beta_1 + (\beta_2 + \beta_3 * c) * t)\right)} + \varepsilon, \tag{4}$$

where β_0 determines asymptotic performance, β_1 determines initial performance, β_2 is the learning rate in the control condition, β_3 measures how much learning is accelerated by feedback ($c = 1$ indicates the presence of feedback), and ε is an i.i.d. random perturbation. To determine whether performance improved with experience and whether this improvement was modulated by feedback we performed a nonlinear regression analysis (`fitnlm` in `Matlab 2015b`) to estimate the coefficients β and test whether β_2 and β_3 were significantly greater than zero. This analysis revealed significant effects of learning on the number of clicks ($t(1197) = 5.04, p < .0001$), the frequency of choosing the optimal

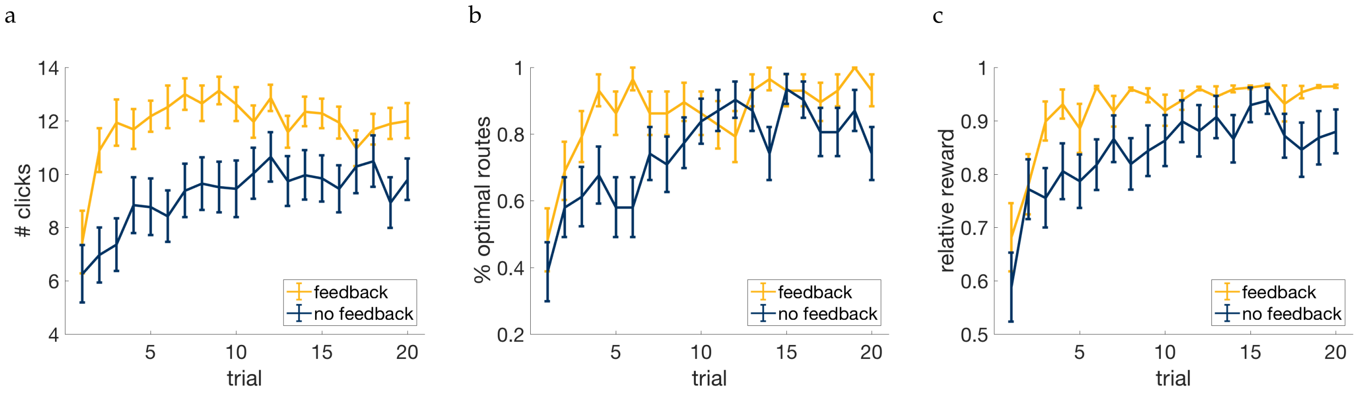


Figure 2: Learning curves for the number of outcomes considered during planning (a), the frequency of identifying the optimal path (b), and relative reward earned (c) with versus without feedback.

route ($t(1197) = 5.35, p < .0001$), and the relative reward ($t(1197) = 5.67, p < 0.0001$) in both conditions. Critically, feedback significantly increased the slope of the sigmoidal learning curves of the proportion of inspected cells ($t(1197) = 2.02, p < .05$), relative performance ($t(1197) = 3.43, p < .001$), and the relative frequency of the optimal route ($t(1197) = 3.29, p < .001$). Model comparisons confirmed that the full model including the feedback regressor explained the data significantly better than the restricted model without it (clicks: $F(1, 1196) = 95.39, p < 10^{-15}$, frequency of optimal solution: $F(1, 1196) = 38.90, p < 10^{-9}$, relative reward: $F(1, 1196) = 33.12, p < 10^{-7}$). This indicates that the feedback we generated from the meta-level pseudo-rewards significantly accelerated the process by which people learned to plan more and better. As a result, participants in the feedback condition learned to consider more possible outcomes ($11.9/16$ vs. $9.1/16$, $F(1, 1160) = 100.38, p < 0.0001$) and consequently earned more relative reward than participants in the control condition (92% vs. 84%, $F(1, 1160) = 43.03, p < .0001$).

4 Conclusion

We have derived a computational method for designing feedback structures that promote metacognitive learning and illustrated its potential to help people learn to plan better. The results presented here build on recent work showing that cognitive plasticity can be understood in terms of meta-level reinforcement learning (Krueger et al., 2017; Lieder, Shenhav, Musslick, & Griffiths, in revision). Together with the preliminary results presented here these findings inspire hope that theory and tools of reinforcement can be leveraged to improve the human mind by promoting metacognitive learning. The results presented here are a first step in evaluating meta-level reward shaping as a theoretical foundation for cognitive training. Future work will rigorously evaluate this feedback mechanism against alternatives, assess retention, and evaluate transfer. If successful this line of research might lead to principled and more effective tools and interventions for training high-level cognitive functions such as planning and decision making, cognitive control, reasoning, problem solving, attention, and learning.

References

Anguera, J. A., Boccanfuso, J., Rintoul, J. L., Al-Hashimi, O., Faraji, F., Janowich, J., … Gazzaley, A. (2013). Video game training enhances cognitive control in older adults. *Nature*, *501*(7465), 97–101.

Bavelier, D., Green, C. S., Pouget, A., & Schrater, P. (2012). Brain plasticity through the life span: learning to learn and action video games. *Annual review of neuroscience*, *35*, 391–416.

Callaway, F., Lieder, F., Krueger, P. M., & Griffiths, T. L. (2017). Mouselab-MDP: A new paradigm for tracing how people plan. In *The 3rd multidisciplinary conference on reinforcement learning and decision making*.

Green, C. S., & Bavelier, D. (2008). Exercising your brain: a review of human brain plasticity and training-induced learning. *Psychology and aging*, *23*(4), 692.

Gureckis, T. M., Martin, J., McDonnell, J., Rich, A. S., Markant, D., Coenen, A., … Chan, P. (2016). psiTurk: An open-source framework for conducting replicable behavioral experiments online. *Behavior research methods*, *48*(3), 829–842.

Hay, N., Russell, S., Tolpin, D., & Shimony, S. (2012). Selecting computations: Theory and applications. In N. de Freitas & K. Murphy (Eds.), *Uncertainty in artificial intelligence: Proceedings of the twenty-eighth conference*. Corvallis, OR: AUAI Press.

Krueger, P. M., Lieder, F., & Griffiths, T. L. (2017). Enhancing metacognitive reinforcement learning using reward structures and feedback. In *Proceedings of the 39th annual conference of the cognitive science society*.

Lieder, F., & Griffiths, T. L. (2016). Helping people make better decisions using optimal gamification. In *Proceedings of the 38th annual meeting of the cognitive science society* (pp. 2075–80). Austin, TX: Cognitive Science Society.

Lieder, F., Shenhav, A., Musslick, S., & Griffiths, T. (in revision). Rational metareasoning and the plasticity of cognitive control.

Morrison, A. B., & Chein, J. M. (2011). Does working memory training work? the promise and challenges of enhancing cognition by training working memory. *Psychonomic bulletin & review*, *18*(1), 46–60.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In I. Bratko & S. Dzeroski (Eds.), *Proceedings of the 16th Annual International Conference on Machine Learning* (pp. 278–287). San Francisco: Morgan Kaufmann.

Owen, A. M., Hampshire, A., Grahn, J. A., Stenton, R., Dajani, S., Burns, A. S., … Ballard, C. G. (2010). Putting brain training to the test. *Nature*, *465*(7299), 775–778.

Russell, S., & Wefald, E. (1991). Principles of metareasoning. *Artificial Intelligence*, *49*(1-3), 361–395.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA, USA: MIT press.

Mouselab-MDP: A new paradigm for tracing how people plan

Frederick Callaway [1]
Dept. of Psychology, UC Berkeley
`fredcallaway@berkeley.edu`

Falk Lieder [1]
Dept. of Psychology, UC Berkeley
`falk.lieder@berkeley.edu`

Paul M. Krueger [1]
Dept. of Psychology, UC Berkeley
`pmk@berkeley.edu`

Thomas L. Griffiths
Dept. of Psychology, UC Berkeley
`tom_griffiths@berkeley.edu`

[1] **These authors contributed equally.**

Abstract

Planning is a latent cognitive process that cannot be observed directly. This makes it difficult to study how people plan. To address this problem, we propose a new paradigm for studying planning that provides experimenters with a timecourse of participant attention to information in the task environment. This paradigm employs the information-acquisition mechanism of the Mouselab paradigm, in which participants click on options to reveal the outcome of choosing those options. However, in contrast to the original Mouselab paradigm, our paradigm is a sequential decision process, in which participants must plan multiple steps ahead to achieve high scores. We release Mouselab-MDP open-source as a plugin for the JsPsych online Psychology experiment library. The plugin displays a Markov decision process as a directed graph, which the participant navigates to maximize reward. To trace the the process of planning, the rewards associated with states or actions are initially occluded; the participant has to click on a transition to reveal its reward. This information gathering behavior makes explicit the states the participant considers. We illustrate the utility of the Mouselab-MDP paradigm with a proof-of-concept experiment in which we trace the temporal dynamics of planning in a simple environment. Our data shed new light on people's approximate planning strategies and on how people prune decision trees. We hope that the release of Mouselab-MDP will facilitate future research on human planning strategies. In particular, we hope that the fine-grained time course data that the paradigm generates will be instrumental in specifying algorithms, tracking learning trajectories, and characterizing individual differences in human planning.

Keywords: planning; process tracing; meta-decision making; pruning; research methods

Acknowledgements

This work was supported by grant number ONR MURI N00014-13-1-0341 and a grant from the Templeton World Charity Foundation.

1 Introduction and Background

Many if not all of us have found ourselves in an unfortunate predicament due to a lapse of forethought. However, despite the ease with which such events come to memory, they are the exception rather than the rule. Humans have an exceptional ability to establish long-term goals and make steady progress towards their completion, often over the course of years or decades. This is only possible through planning. Planing is a broad notion, potentially taking multiple forms (Morris & Ward, 2004); however for present purposes, we define planning as explicitly considering potential future states, the actions one might take in them, the states that might result from those actions, and so on. For example, a student deciding whether or not to attend a class, is engaged in planning when they consider doing homework during class time and skimming the slides on the morning of the following lecture.

Planning is a fundamental aspect of higher-order cognition, and it has accordingly received much attention in cognitive psychology. Planning has been studied with verbal protocol analysis and computer simulation of problem solving (Newell, Simon, et al., 1972), models of the hierarchical structure of human behavior (Miller, Galanter, & Pribram, 1986; Botvinick, Niv, & Barto, 2009), errors and reaction times in sequential decision-making (e.g., Huys et al., 2012, 2015), and neural activity in spatial navigation tasks (Ólafsdóttir, Barry, Saleem, Hassabis, & Spiers, 2015; Simon & Daw, 2011; Balaguer, Spiers, Hassabis, & Summerfield, 2016). Planning has also received considerable attention in artificial intelligence (Russell, Norvig, & Intelligence, 2010; LaValle, 2006), and Markov decision processes (Sutton & Barto, 1998) have emerged as a common mathematical framework for bridging these two literatures.

Research on planning is complicated by the fact that we cannot directly observe the cognitive processes of planning. Thus, researchers must infer this latent cognitive process based on the decisions participants ultimately make, or on neural data which are challenging to collect and interpret. One approach to this problem is to design *process tracing* paradigms that externalize some aspect of the cognitive process. Payne, Bettman, and Johnson (1988) developed one such methodology for studying multi-alternative risky choice: the “Mouselab” paradigm. The paradigm presents a decision problem as a payoff matrix whose entries are initially occluded. Participants choose between multiple gambles whose outcome-dependent payoffs are recorded in the cells of the payoff matrix. Each column of the matrix corresponds to a gamble and each row of the matrix corresponds to one of the possible outcomes. The outcome probabilities are known and the gambles differ only in how much money they assign to each of the possible outcomes. Critically, to find out how much money a gamble pays for each of the outcomes, the participant has to click on the corresponding cell of the payoff matrix. This series of clicks makes experimentally observable the information a participant considers in making a decision. This new form of data led to novel insights into people’s decision processes including the discovery of adaptive strategy selection (Payne et al., 1988).

Although the traditional Mouselab paradigm is very useful for studying decision making, it cannot be used to study planning because the decision on one gamble does not affect future gambles. Thus, to apply the Mouselab process tracing method to planning, we simply replace the single decision with a Markov Decision Process (MDP), in which a participant must make a sequence of choices, each one affecting the choices that will be available in the future. By releasing the paradigm open-source as a JsPsych plugin (De Leeuw, 2015), we hope to make it possible for experimenters to form detailed analyses of planning processes without extensive programming or expensive equipment.

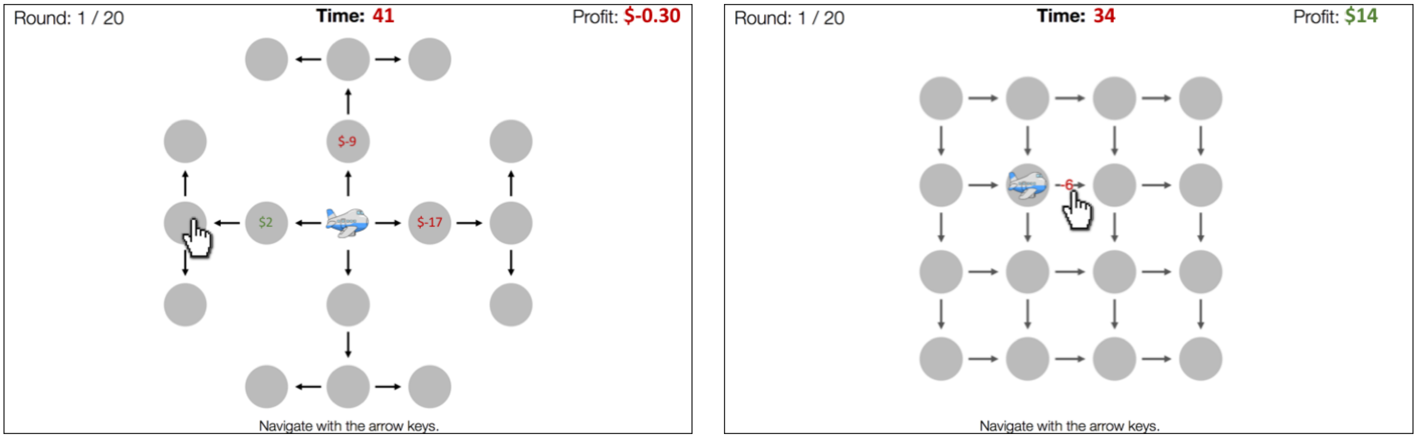
2 Using Mouselab-MDP

Having motivated the paradigm, we briefly describe both the interface through which experimenters specify experiments, and the interface through which participants engage in the task. Two screenshots of the paradigm are shown in Figure 1, and a live demo can be viewed at http://cocosci.dreamhosters.com/webexpt/mouselab-demo/. The code for Mouselab-MDP lives at https://github.com/fredcallaway/Mouselab-MDP.

On each trial an MDP is presented with an intuitive spatial interface: a directed graph with states as nodes and actions as edges. The participant navigates through the graph using the keyboard, attempting to collect the maximal total reward. States or edges are annotated with the reward for reaching the state or taking the action; but crucially, these labels may not be visible when the trial begins. The participant may need to click or hover the mouse over a state/edge to see the associated reward; the timecourse of these information-gathering actions provides fine-grained evidence of the planning process. Furthermore, there may be a price associated with gathering information; this creates a tradeoff between the cost and value of information, which is the focus of work on bounded optimality and resource rationality (Griffiths, Lieder, & Goodman, 2015; Shenhav et al., 2017).

With the Mouselab-MDP plugin you can create a planning experiment by specifying the following critical components:

1. `graph` is a mapping $s \mapsto A$ where A is itself a mapping $a \mapsto (r, s')$. This structure specifies the actions a available in each state, as well as the reward r and resultant state s' associated with each action.
2. `initial` is the state in which the participant begins the trial. Along with `graph`, this defines a deterministic MDP.



a) State values revealed with clicks

b) State values shown while hovering the mouse

Figure 1: Two example paradigms created with the Mouselab-MDP plugin for JsPsych: a) Each state is labeled with the reward for reaching that state; these rewards become visible after they are clicked, with a $0.10 fee per click. b) The reward for making a transition is revealed only while the mouse is hovering over the corresponding arrow.

3. `layout` is a mapping $s \mapsto (x, y)$ that specifies the location of each state on the screen.

Specifying only these settings will result in a graph with rewards shown on the edges between nodes and no labels on the states. This corresponds to a standard MDP with a known transition and reward function. To take advantage of the Mouselab features, the user must specify at least one of the following optional properties:

1. `stateLabels` is a mapping $s \mapsto \ell$ that specifies the labels to be shown on each state.
2. `stateDisplay` $\in$ { 'never', 'hover', 'click', 'always' } specifies when state labels are displayed. When set to 'click', clicking on the state causes the label to appear and remains on the state for the duration of the trial. The optional parameter `stateClickCost` specifies the cost (a negative number) for clicking on a single state. When set to 'hover', the label appears only while the mouse is hovering over the associated edge. There is no cost for this option due to the likelihood of accidentally passing the mouse over an edge.
3. `edgeLabels` is analagous to `stateLabels`, except that it defaults to the rewards associated with each edge.
4. `edgeDisplay` is analagous to `stateDisplay`. `edgeClickCost` specifies the cost.

This novel paradigm offers three advantages over previous behavioral paradigms for the study of planning:

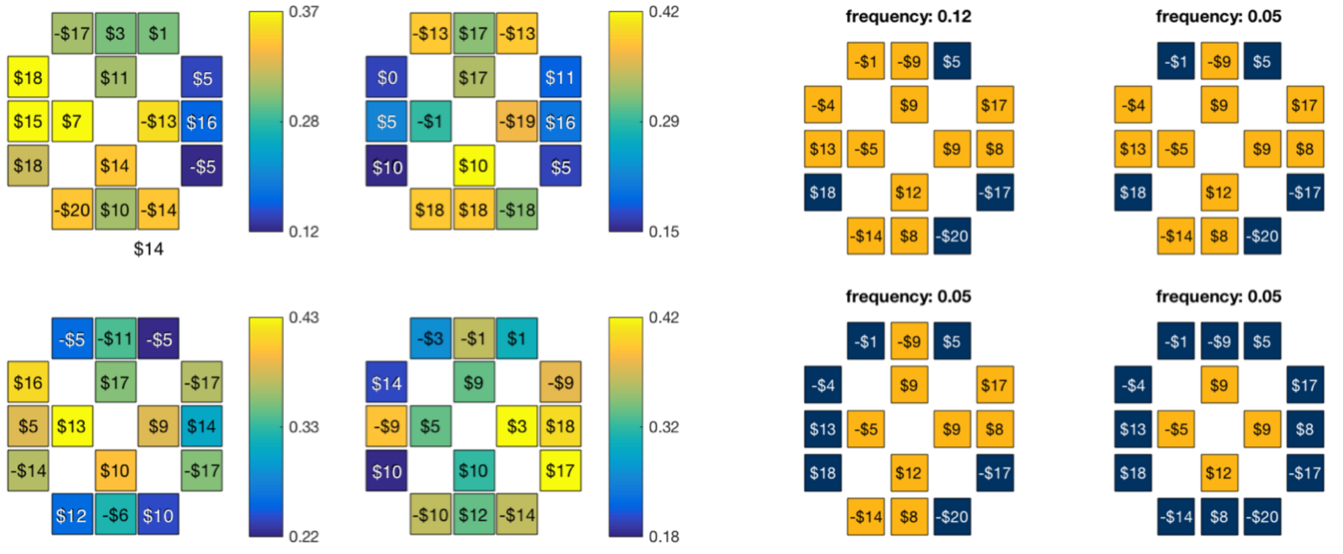
1. The latent cognitive process of planning is made observable by recording the participant's clicks. This allows the experimenter to test hypotheses that would otherwise be difficult to confirm or refute based on overt behavior.
2. A wide variety of state-transition and reward structures can be displayed automatically. This allows an experimenter to create a large number of highly variable stimuli (potentially automatically with a computer program) in relatively little time (compared to using an image-editing software).
3. The paradigm is packaged as a JsPsych plugin with a concise, yet flexible stimuli specification. This allows experimenters with only basic knowledge of Javascript to use our plugin to create a wide range of qualitatively novel experiments that can be run online with crowd-sourcing services such as Amazon Mechanical Turk.

3 Revealing Planning Strategies

Here, we present a proof-of-concept case study to illustrate the utility of the Mouselab-MDP paradigm. The data from this experiment sheds new light on the well-studied notion of pruning (Huys et al., 2012).

3.1 Methods

We used the Mouselab-MDP plugin for JsPsych. In each trial participants route an airplane from the center of the screen to one of eight final destinations via two intermediate locations (see Figure 1a). We specified `stateLabels` as the rewards associated with the edge leading to each state. We set `stateDisplay` to 'click' and 'stateClickCost' to 0.10; thus participants could click on a state to reveal the reward for traveling to that state, at the price of $0.10. Participants



a) Frequency of sampling by location, across four MDPs.

b) The four most common sets of clicks for one MDP.

Figure 2: Patterns of mouse clicking in humans. a) Each subplot shows the location and monetary value of each state for a different MDP. The colors indicate how often each state was inspected before the first move. b) Frequencies indicate the portion of participants who selected each set, for one particular MDP. Inspected states are gold. The monetary values indicate the reward for flying to that state.

were required to spend at least 45 seconds on every trial to prevent time cost from discouraging participants from clicking and planning.

We recruited 31 participants on Amazon Mechanical Turk. The task took 31 minutes on average and participants were paid $\$1.50$ plus a bonus equal to 5% of their earnings on one randomly selected trial (average bonus: $\$1.69$). Participants completed a total of 20 trials in randomized order. The trials differed only in rewards. All rewards were integers between $-\$20$ and $+\$18$ (mean: 4.2, SD: 10.3). The reward functions were designed so that 3-step planning with full information led to significantly higher returns than 2-step planning, 1-step planning, or random choice ($\$38.35$ vs. $\$16.06$–$\$18.30$).

3.2 Results

Figure 2a shows the frequency with which participants inspected each of the sixteen states of the task environment (Figure 1a) prior to selecting the first flight, in four of the 20 MDPs. Each MDP corresponds to a decision tree with four branches each of which comprises a stem and three outer nodes. Examining Figure 2a (and similar plots for all the MDPs), the frequency with which participants inspected the outer nodes of a branch appears to increase with the reward at the stem of the branch. Conversely, participants were less likely to inspect the outer nodes of a branch when its inner node was a large loss. This is indicative of pruning. To quantify this intuitive result, we define pruning as inspecting none of the subsequent nodes of a branch after inspecting the stem. Figure 3 shows that the probability with which people prune a branch of their decision tree increases gradually with the magnitude of the loss at its stem. A logistic regression analysis accounting for individual differences in the base rate of pruning confirmed that the increase in pruning with the decrease of the reward at the stem of the branch was statistically significant ($t(1276) = -2.01, p < 0.05$). To our knowledge, this is the first analysis to find that pruning changes gradually with the magnitude of the loss or gain.

Our results are consistent with those of Huys et al. (2012) who showed that a model with a "specific pruning parameter" (i.e. a larger discount γ applied to states after a large loss) fit participant choices better than a model without such a parameter. However, their data and model are also consistent with an alternative explanation: people may consider all branches of the tree but discount distant rewards while overweighting immediate losses (Green & Myerson, 2004). Our data, in contrast, provide direct evidence that participants did not consider states after costly transitions. This indicates that the continuous discounting parameter of Huys et al. (2012) may have captured the average effect of many discrete pruning decisions.

Figure 2b shows the four most common sets of clicks (before the first flight) for a one MDP. The most frequent strategy inspected exactly one complete path on each of the four branches, and the second most common strategy pruned one or more branches whose reward at the stem was lowest.

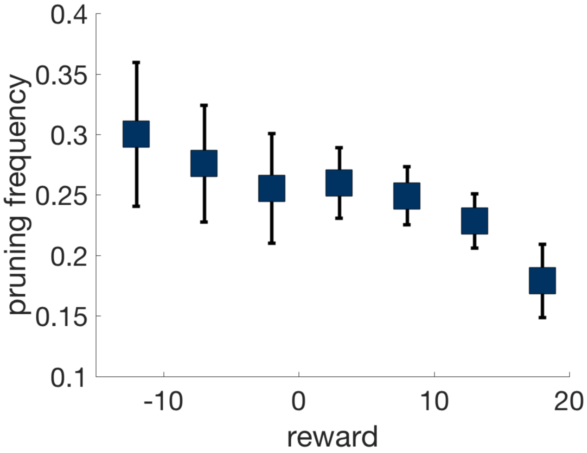


Figure 3: Frequency of pruning ±1SEM (vertical axis) given the reward at the stem of the branch (horizontal axis). Pruning was defined as inspecting none of a branch's outer states (before the second move) after having inspected the reward at its stem (before the first move). Each point shows the mean frequency of pruning.

4 Conclusion

We have developed a process-tracing paradigm for the study of planning and demonstrated that it can reveal the processes of planning at a higher level of resolution than standard behavioral paradigms. We showed that the paradigm provides a new look at one aspect of planning: pruning. However, many questions remain. Do people plan in a breadth-first or depth-first fashion? Do they plan and then act all at once, or do they interleave planning and acting? How do they balance the cost of planning with the potential for increased reward? Do these aspect of planning vary based on the problem structure? If so, how do people meta-reason about what planning strategies to employ? How do people learn and refine their planning strategies? How and why do people differ in their planning abilities? No single paradigm can give an answer to all these questions. However, Mouselab-MDP can produce data that speaks to all of them.

Future work will extend the Mouselab-MDP plugin to stochastic environments and enable tracing additional aspects of the planning process, including representation of a transition model and heuristic evaluation of state values.

References

Balaguer, J., Spiers, H., Hassabis, D., & Summerfield, C. (2016). Neural mechanisms of hierarchical planning in a virtual subway network. *Neuron*, *90*(4), 893–903.

Botvinick, M. M., Niv, Y., & Barto, A. C. (2009). Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, *113*(3), 262–280.

De Leeuw, J. R. (2015). jspsych: A javascript library for creating behavioral experiments in a web browser. *Behavior Research Methods*, *47*(1), 1–12.

Green, L., & Myerson, J. (2004). A discounting framework for choice with delayed and probabilistic rewards. *Psychological bulletin*, *130*(5), 769.

Griffiths, T. L., Lieder, F., & Goodman, N. D. (2015). Rational use of cognitive resources: Levels of analysis between the computational and the algorithmic. *Topics in cognitive science*, *7*(2), 217–229.

Huys, Q. J., Eshel, N., O'Nions, E., Sheridan, L., Dayan, P., & Roiser, J. P. (2012). Bonsai trees in your head: how the Pavlovian system sculpts goal-directed choices by pruning decision trees. *PLoS Comput Biol*, *8*(3), e1002410.

Huys, Q. J., Lally, N., Faulkner, P., Eshel, N., Seifritz, E., Gershman, S. J., … Roiser, J. P. (2015). Interplay of approximate planning strategies. *Proceedings of the National Academy of Sciences*, *112*(10), 3098–3103.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

Miller, G. A., Galanter, E., & Pribram, K. H. (1986). *Plans and the structure of behavior.* Adams Bannister Cox.

Morris, R., & Ward, G. (2004). *The cognitive psychology of planning*. Psychology Press.

Newell, A., Simon, H. A., et al. (1972). *Human problem solving* (Vol. 104) (No. 9). Prentice-Hall Englewood Cliffs, NJ.

Ólafsdóttir, H. F., Barry, C., Saleem, A. B., Hassabis, D., & Spiers, H. J. (2015). Hippocampal place cells construct reward related sequences through unexplored space. *Elife*, *4*, e06063.

Payne, J. W., Bettman, J. R., & Johnson, E. J. (1988). Adaptive strategy selection in decision making. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *14*(3), 534.

Russell, S., Norvig, P., & Intelligence, A. (2010). *Artificial intelligence: A modern approach* (3rd ed., Vol. 25). Prentice-Hall, Englewood Cliffs.

Shenhav, A., Musslick, S., Lieder, F., Kool, W., Griffiths, T. L., Cohen, J. D., & Botvinick, M. M. (2017). Toward a rational and mechanistic account of mental effort.

Simon, D. A., & Daw, N. D. (2011). Neural correlates of forward planning in a spatial decision task in humans. *Journal of Neuroscience*, *31*(14), 5526–5539.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1) (No. 1). Cambridge, MA: MIT press.

Learning to (mis)allocate control: maltransfer can lead to self-control failure

Laura Bustamante *
Princeton Neuroscience Institute
Princeton University
Princeton, NJ
lauraab@princeton.edu

Falk Lieder *
Helen Wills Neuroscience Institute
University of California, Berkeley
falk.lieder@berkeley.edu

Sebastian Musslick
Princeton Neuroscience Institute
Princeton University
Princeton, NJ
musslick@princeton.edu

Amitai Shenhav
Cognitive, Lingusitic, & Psychological Science
Brown Institute for Brain Sciences
Brown University
Providence, RI
amitai_shenhav@brown.edu

Jonathan Cohen
Princeton Neuroscience Institute
Department of Psychology
Princeton University
Princeton, NJ
jdc@princeton.edu

* **These authors contributed equally.**

Abstract

How do people learn when and how much control to allocate to which cognitive mechanism? A satisfactory answer to this question should account not only for people's adaptive control strategies but also for common forms of self-control failure including the phenomenon that people sometimes engage in effortful controlled processing even when it harms performance relative to automatic alternatives. For example, a driver who focuses so much of their attention on solving a complex math problem that they fail to notice the traffic ahead of them. We propose that people transfer what they have learned about the value of control in a particular situation to other situations with similar features and formally express this in a computational model. We explore whether failures of self-control may result from maltransfer in learning a rational approximation of the optimal control policy prescribed by the Expected Value of Control theory. We designed a novel color-word Stroop paradigm where reward for a task performed on an incongruent stimulus is jointly determined by the color and meaning of the word. In an initial "association phase" words and colors were reinforced for performing either color-naming (CN) or word-reading (WR). In a "transfer phase" CN was rewarded when either the word or the color were previously associated with it (SINGLE trials) but when both the word and the color were associated with CN the correct response was WR (X trials). We vary the frequency of SINGLE trials from 0% to 50% and hypothesize participants would incorrectly transfer the control demand they experienced on SINGLE trials to X trials and consequently reduce their reward rate. Empirical data from 30 participants confirmed this hypothesis and supports the conclusion that maltransfer in learning about the value of control can mislead people to overexert cognitive control even when it hurts their performance.

Keywords: cognitive control; cognitive plasticity; metacognitive reinforcement learning; self-control failure

Acknowledgements

This work was made possible by the John Templeton Foundation, grant number 57876, and the National Institutes of Health, grant number T32MH065214. The opinions expressed in this publication are those of the authors and do not necessarily reflect the views of the John Templeton Foundation.

1 Introduction and background

Cognitive control is the collection of mechanisms that bias the cognitive system away from a default processing policy in the service of task-relevant goals. This is critical to everyday functioning and encompasses self-control, the ability to override automatic response tendencies, attention, working memory, and cognitive flexibility (*1*). Exerting too little cognitive control can have devastating consequences when the default (automatic) response is maladaptive. But the overexertion of cognitive control can be just as problematic, for instance when a driver fails to handle routine traffic situations because they compulsively allocated too much cognitive resources to discussing a RLDM abstract with their co-driver. The overexertion of cognitive control is a curious phenomenon as it entails the effortful allocation of cognitive control despite detrimental consequences.

Our model builds on the expected value of control (EVC) theory by (*2*). The EVC theory provides a normative account of how the brain allocates its limited capacity for controlled processing (*3–5*) between competing processes. According to the EVC theory, control signals should be selected according to a rational cost-benefit analysis that computes the expected value of the reward gained by exerting cognitive control minus its cost. This normative prescription poses a considerable computational challenge as computing the exact value of the EVC across a high-dimensional space of control signals would generally exceed available time and cognitive resources (*6*). To address this challenge, recent work has proposed that the brain learns an efficient approximation of the expected value of control (*7*). To be efficient this learning mechanism has to assume that the expected value of control is a linear, additive function of relevant features of the control signals and the current situation. Critically, there are real-world scenarios that violate this assumption. For instance, while it is valuable to allocate cognitive control to focus one's thoughts on a topic under discussion, and it is also valuable to allocate attention to the road while driving, the value of doing both at the same time is significantly less than the sum of their values. The proposed learning mechanism might therefore be able to explain how people come to allocate too much control to processes that harm their performance The following sections formalize this hypothesis and test its predictions empirically.

2 The Expected Value of Control Model of the Plasticity of Cognitive Control

How does the brain learn when to exert cognitive control, which control signal to specify, and how much control to exert? According to the EVOC theory, the dorsal ACC learns to predict the EVC of specifying the control signal c in situation s (EVC(s,c)) from their features $\mathbf{f}(s,c)$, that is

$$\text{EVC}(s,c) \approx \hat{q}(s,c;w) = \sum_i w_i \cdot f_i(s,c), \tag{1}$$

where w_i is the weight of the ith feature. For instance, for a Stroop task with the colors red and green and the words 'green' and 'red', the features would include $f_1(s,c) = \text{colorIsRed}(s) \cdot c$, $f_2(s,c) = \text{colorIsGreen}(s) \cdot c$, $f_3(s,c) = \text{wordIsRed}(s) \cdot c$, $f_4(s,c) = \text{wordIsGreen}(s) \cdot c$ as well as $f_5(s,c) = \text{cost}(s,c)$, and $f_6(s,c) = c$.

According to the the EVOC model, the feature weights $\mathbf{w} = (w_1, \cdots, w_5)$ are learned by Bayesian linear regression of the experienced value of control $\hat{Q}(s,c) = R - \text{cost}(s,c)$ onto the features $\mathbf{f}$, where R is the reward experienced upon control exertion and $\text{cost}(s,c)$ is the control cost. The resulting posterior distribution $P(\mathbf{w}|E)$ over the weights given the agent's experience E is then used to select the control signal $c^\star$ via Thompson sampling, that is

$$c^\star = \arg\max_c \hat{q}(s,c;\tilde{w}), \text{ where } \tilde{w} \sim P(\mathbf{w}|E). \tag{2}$$

Here, we apply this theory to predict people's accuracy and reaction times in a novel color-word Stroop task. Following (*6*), we derive these predictions from the selected control signals via a drift-diffusion model with the drift rate

$$d = c^\star \cdot y_{\text{color}} \cdot d_{\text{color naming}} + \cdot(1 - c^\star) \cdot y_{\text{color}} \cdot d_{\text{word reading}}, \tag{3}$$

where $y_{\text{color}}, y_{\text{word}} \in \{-1, 1\}$ are the responses reporting the color respectively the word, and $d_{\text{color naming}} = 3.55$ and $d_{\text{word reading}} = 10$ are the drift rates of the word reading process and the color naming process respectively[1].

The key idea for the purpose of the present study is that the value of control is approximated by a linear combination of situational features such as a particular instance of a word presented in a Stroop task and its color (and a more detailed description of the the model can be found in Lieder, Shenhav, Musslick, & Griffiths, in revision). This account predicts transfer effects between different stimuli that share common features. For instance, if the stimulus GREEN demands cognitive control, then exposure to it will increase the weight w_1 of the feature $f_1(s,c) = \text{colorIsRed}(s) \cdot c$ and the weight w_2 of the feature $f_4(s,c) = \text{wordIsGreen}(s,c) \cdot c$. For the resulting weights w_2' and w_4' the predicted value of control $\hat{q}(\text{GREEN}, c; \mathbf{w}')$ will increase more with increasing control intensity c; consequently the optimal control intensity

---

[1] The drift rates for color naming and word reading reflect their relative default processing strength (*8*).

$c^\star$ will be higher than it was for the original weight vector $\mathbf{w}$. Critically, to a lesser extent the same will happen for other red words, such as BLUE, and instances of the word GREEN that are written in a different color, such as GREEN. Such transfer effects are beneficial when stimuli that share features also share similar demands for cognitive control. Yet, such transfer effects can also be detrimental when stimuli that share features differ in their demands for cognitive control. Concretely, if the stimulus GREEN demands color naming, whereas the stimuli GREEN and BLUE demand word reading, then repeated exposure to GREEN should interfere with subsequent responses to GREEN and BLUE by misleading participants to overexert control where the automatic response would have been correct. Therefore, the EVOC model predicts that exposure to control-demanding stimuli will interfere with people's responses to automaticity-demanding stimuli that share some of their features. Interference should manifest by decreased accuracy despite longer reaction times and the magnitude of the interference effect should increase with the frequency of the control-demanding stimuli. The following experiment uses a novel cognitive control task to test the model's prediction that learning in control demanding situations results in maltransfer to similar situations where the automatic response is correct.

3 Empirical test of maltransfer with the XOR-Stroop task

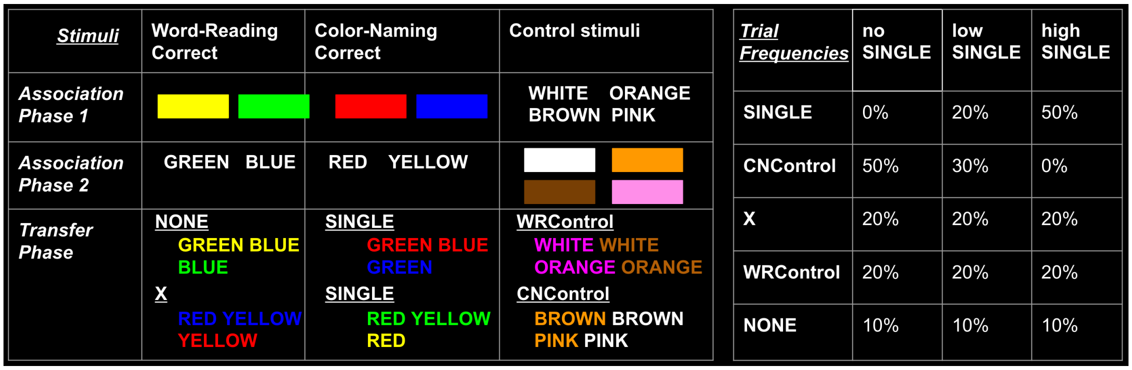
3.1 Methods

30 participants at Princeton University (16 female, 18 - 53 years old, mean = 22) completed a color-word Stroop task. Subjects are instructed that they may respond to to either the word (word reading, WR) or the color the word is written in (color naming, CN), all trials were incongruent. Subjects learn through experience which response is rewarded for unique stimuli. After giving a response by keypress subjects receive as feedback a number indicating how much reward was on offer for that trial (e.g. +10) and whether they obtained that reward (**e.g. +10**). Participants received one dollar of bonus compensation for every 200 points (bonus compensation ranged from 10 to 17 dollars, mean = 13 dollars).

The correct response (CN or WR) was determined by a conjunction of features, the ink color identity and the word identity. We hypothesized that if a color identity has been rewarded for the CN response (red and blue) and a word identity has been rewarded for CN (**RED** and **YELLOW**), then when subjects encounter these in combination (**RED**, **YELLOW**, or **YELLOW**) they would CN even after experience that this was incorrect. These trials are referred to as 'X' trials and are the focus of our analyses. In two association phases we associate our colors and words of interest with either CN or WR response. In the transfer phase we present all possible incongruent combinations of our colors and words of interest and manipulate their relative frequencies to examine transfer effects across stimuli (see Figure 1).

Stimuli: One subset of the 8 colors and words (yellow/green/red/blue, **YELLOW**/ **GREEN**/ **BLUE**/ **RED**) were of interest, the other subset (white/orange/brown/pink, **WHITE**/ **ORANGE**/ **BROWN**/ **PINK**) were used as controls. Subjects first learn key press mappings for 8 total possible colors and words. Subjects performed trials in three 'phases' with self-timed breaks in between, subjects were not cued to any differences between the phases. In *Association Phase 1*, we create associations with four colors of interest (combined with control words) and a response. For example every time the ink color yellow was presented the correct response was WR. In *Association Phase 2* we create associations with our words of interest (in combination with control colors) and a response, for example every time the word YELLOW was presented the correct response was CN. Stimuli in the *Transfer Phase* are described by six categories. In the case of "SINGLE" trials, a single feature has been rewarded for CN during association training (whereas the other has been rewarded for WR) and CN is the correct response. For "NONE" trials neither features have been rewarded for CN, and WR is correct. For X trials, both features have been rewarded for CN, however the CN response is incorrect. We also include "CNControl" trials where a subset of the control colors and control words are combined and CN is the correct response and "WRControl" trials with a separate subset of control stimuli combinations where WR is correct. Participants completed 520 total trials, 160 trials in each Association Phase and 200 trials at Transfer Phase. During association phases subjects were rewarded 5 points for each correct response, during transfer phase subjects were rewarded 10 points for each correct response.

SINGLE Frequency Manipulation: Because SINGLE stimuli have shared features with X and NONE stimuli we predict that increasing SINGLE trial frequency (and accordingly reward resulting from these trials) will increase propensity to CN on X and NONE trials. We tested three SINGLE frequencies in a between-subjects design (n = 10 subjects per condition), subjects were not informed about stimulus frequencies. In the "no SINGLE" condition subjects were presented only X, NONE, CNControl, and WRControl. In the "low SINGLE" condition 20 percent of trials were SINGLE trials, and in the "high SINGLE" 50 percent of trials were SINGLE trials. We use CNControl trials to ensure CN is correct on one half of all trials regardless of frequency condition group. WRControl trials occur at the same frequency as X trials but share no features with SINGLE trials and therefore allow us to evaluate the contribution of feature-specific transfer effects relative to global control adjustments (i.e. higher tendency to name the color irrespective of the stimulus). In all conditions the frequency of X and WRControl trials was 20 percent and NONE trials comprised 10 percent of total trials in each condition (see figure 1).



Stimuli	Word-Reading Correct	Color-Naming Correct	Control stimuli
Association Phase 1	🟨 🟩	🟥 🟦	WHITE ORANGE BROWN PINK
Association Phase 2	GREEN BLUE	RED YELLOW	⬜ 🟧 🟫 🟪
Transfer Phase	NONE GREEN BLUE BLUE X RED YELLOW YELLOW	SINGLE GREEN BLUE GREEN SINGLE RED YELLOW RED	WRControl WHITE WHITE ORANGE ORANGE CNControl BROWN BROWN PINK PINK

Trial Frequencies	no SINGLE	low SINGLE	high SINGLE
SINGLE	0%	20%	50%
CNControl	50%	30%	0%
X	20%	20%	20%
WRControl	20%	20%	20%
NONE	10%	10%	10%

Figure 1: Stimuli and experimental design.

3.2 Model predictions

For this specific experimental design our model predicted that the frequency of SINGLE trials should significantly decrease participants' accuracy on X trials and NONE trials and remain consistently higher on NONE trials than on X trials. Furthermore, the model also predicts a V-shaped relationship between the frequency of SINGLE trials and participants' reward rate: the addition of 25% SINGLE trials should significantly decrease the reward rate due to the maltransfer from SINGLE trials to X trials and NONE trials. Yet, increasing the frequency of SINGLE trials further to 50% should increase the reward rate because participants performance on the SINGLE trials should be high and adding more trials on which participants perform well increases their returns. In the following section, we evaluate these predictions against human performance.

3.3 Results

Performance during the *Transfer Phase* was analyzed for all trial types and across frequency groups. A multi-variate analysis of variance (MANOVA) was conducted to compare the effect of SINGLE frequency group and Trial Type on accuracy for our three groups. There was a significant effect of effect of SINGLE frequency group (F(2,135) = 48.062, $p < 0.001$), Trial Type (F(2,135) = 9.517, $p < 0.001$), and their interaction (F(7,135) = 4.524, $p < 0.001$).

X Trials: The association phase was designed to impair performance on X trials at *Test*. The frequency of SINGLE trials was hypothesized to further decrease performance on X trials due to reinforcement of the control-demanding response for features shared with X trials, where the automaticity-demanding response was required. Post-hoc comparisons using Tukey's honest significant difference test showed that accuracy on X trials significantly decreased in the high SINGLE condition as compared to the no SINGLE condition (mean accuracy difference = -29%, $p < 0.001$). Experience during *Training* was predicted to impair performance on X trials but not NONE trials in the absence of SINGLE trials. Post-hoc tests did not reveal performance differences on NONE compared to X trials in the no SINGLE group.

NONE Trials: Features from the NONE trials were reinforced for the automatic-demanding response and so should not interfere with NONE performance at *Transfer*. The frequency of SINGLE trials on the other hand was hypothesized to decrease performance on NONE trials due to shared-features. Compared to no SINGLE, NONE trial accuracy decreased in low SINGLE (mean = -31%, $p < 0.001$) and high SINGLE (mean accuracy difference = -37%, $p < 0.001$).

Control Trials: Since WRControl trials did not share features with SINGLE trials, changes in accuracy on WRControl trials with SINGLE frequency are proposed to reflect non-feature-specific changes in propensity to exert control at *Transfer*. Accuracy on WRControl trials in the high SINGLE condition decreased as compared to the no SINGLE condition (mean accuracy difference = -28%, $p < 0.001$), whereas it did not distinguish error rate between low SINGLE and no SINGLE or high SINGLE. In the case of both SINGLE trials and CNControl trials, non-feature-specific changes in propensity to exert control should either not change, or benefit performance. Consistent with modelling predictions no change in accuracy was observed for CNControl trials or SINGLE trials across SINGLE frequency groups.

Reward Rate: Maltransfer from SINGLE trials to X and NONE trials was predicted to decrease participant reward rate. An MANOVA was conducted to examine the effect of SINGLE frequency condition on mean reward rate (points per second of reaction times). Reward rate significantly decreased from no SINGLE compared to low and high SINGLE frequency (ANOVA, F(2,27) = 4.99, $p < 0.001$). Modelling predicted a recovery of reward rate in the high SINGLE condition however post-hoc test showed no change in reward rate from low to high SINGLE.

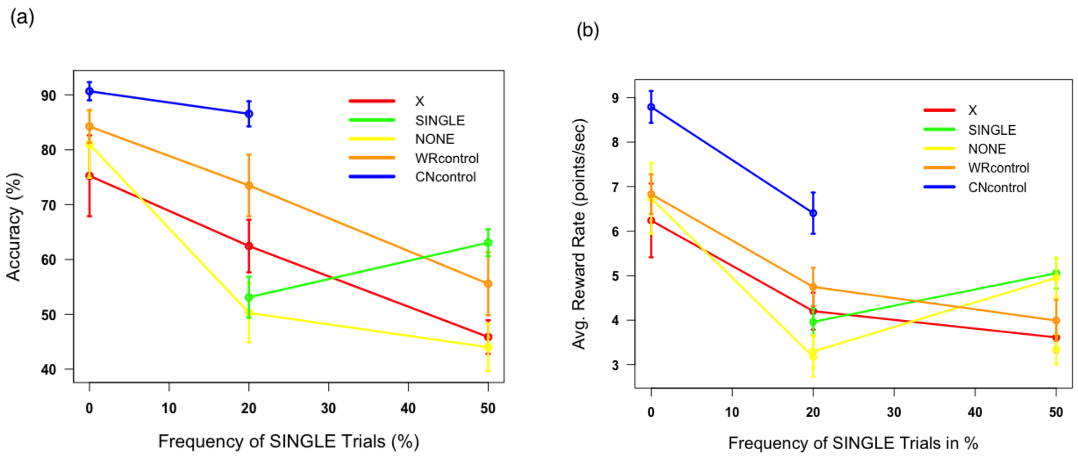


Figure 2: a) Mean accuracy for X, NONE, and WRControl trials decrease as SINGLE trial frequency increases. and b) Reward rate decreases as SINGLE trial frequency increases.

4 Discussion and Conclusions

In the no SINGLE group subjects performed at only 75% $(7.4\%$ s.e.m.) accuracy on X trials even though the task was to read the word. This remarkably low performance is most likely a consequence of maltransfer from the association phases to the test phase. To examine maltransfer effects further, we parametrically varied the frequency of SINGLE trials. In line with our model's predictions, the addition of SINGLE trials further decreased performance on X trials. Decreases in accuracy did not appear to be driven by a speed-accuracy trade-off as reaction time generally grew as error rates grew, and jointly these resulted in the decreased reward rate observed. The maltransfer from SINGLE trials to NONE trials and X trials appears to have interfered with the performance in the conditions with SINGLE trials. Consistent with this explanation, participants in the conditions with SINGLE trials were not only less accurate but also slower. Consequently, their reward rate was lower. The finding of control-overexertion is particularly striking because people generally avoid the cost and effort associated with cognitive control. The finding that WRControl trials decrease as SINGLE frequency increases suggests that people also learn a global (feature-unspecific) value of exerting control. Additional model-fitting analyses will examine the contribution of different forms of learning over the time-course of the experiment, as well as differences in learning dynamics between individuals.

Our empirical findings support the hypothesis that adjustments of control intensity are driven by a learning mechanism that approximates the EVC by a linear combination of relevant features such as the colors and words in a Stroop task. While this learning mechanism usually results in adaptive improvements in control specification, it leads to maltransfer when situations that share relevant features differ in their demands for cognitive control. We have demonstrated this by constructing a relationship that radically violates the learning mechanisms assumption of additivity. Critically, there are many real-world examples of non-linearities in the value of cognitive control–most importantly multi-tasking–where the simultaneous specification of two valuable control signals (e.g., $c_1 =$ drive and $c_2 =$ sketchEVOCModel) has a significantly lower expected value than either of them alone. This suggests that the proposed cognitive plasticity mechanism might be able to explain how people come to maladaptively over-exert cognitive control.

References

1. A. Diamond, *Annual review of psychology* **64**, 135 (2013).
2. A. Shenhav, M. M. Botvinick, J. D. Cohen, *Neuron* **79**, 217 (2013).
3. M. I. Posner, C. R. Snyder, *Information Processing and Cognition: The Loyola Symposium* (RL Solso, 1975), pp. 55–85.
4. R. Shiffrin, W. Schneider, *Psychological Review* **84**, 127 (1977).
5. D. Navon, D. Gopher, *Psychological Review* **86**, 214 (1979).
6. S. Musslick, A. Shenhav, M. M. Botvinick, J. D. Cohen, *2nd Multidiscip. Conf. Reinf. Learn. Decis. Mak.* (2015).
7. F. Lieder, A. Shenhav, S. Musslick, T. Griffiths (in revision).
8. J. R. Stroop, *Journal of experimental psychology* **18**, 643 (1935).

Helping people choose subgoals with sparse pseudo rewards

Frederick Callaway
Dept. of Psychology, UC Berkeley
`fredcallaway@berkeley.edu`

Falk Lieder
Dept. of Psychology, UC Berkeley
`falk.lieder@berkeley.edu`

Thomas L. Griffiths
Dept. of Psychology, UC Berkeley
`tom_griffiths@berkeley.edu`

Abstract

Many decisions require planning multiple steps into the future, but optimal planning is computationally intractable. One way people cope with this problem is by setting subgoals, suggesting that we can help people make better decisions by helping them identify good subgoals. Here, we evaluate the benefits and perils of highlighting potential subgoals with pseudo-rewards. We first show that sparse pseudo-rewards based on the value function of a Markov decision process (MDP) lead a limited-depth planner to follow the optimal policy in that MDP. We then demonstrate the effectiveness of these pseudo-rewards in an online experiment. Each of 88 participants solved 40 sequential decision-making problems. In control trials, participants only saw the state-transition diagram and the reward structure. In experimental trials, participants additionally saw pseudo-rewards equal to the value (sum of future rewards) for the states 1-, 2-, or 3-steps ahead of the current state. When participants reached one of those states, the display would again reveal the values of the states located 1-, 2-, or 3-steps ahead of the current state. We found that showing participants the value of proximal states induced goal-directed planning and improved their average score per second. This benefit was largest when the incentives were 1 or 2 steps away and decreased as they were moved farther into the future. Although these pseudo-rewards were beneficial overall, they also caused systematic errors: Participants sometimes neglected the costs and rewards along the paths to potential subgoals, leading them to make "unwarranted sacrifices" in the pursuit of the most valuable highlighted states. Overall, our results suggest that highlighting valuable future states with pseudo-rewards can help people make better decisions. More research is needed to understand what constitutes optimal subgoals and how to better assist people in selecting them.

Keywords: pseudo-rewards; shaping; planning; goals; gamification

Acknowledgements

This work was supported by grant number ONR MURI N00014-13-1-0341 and a grant from the Templeton World Charity Foundation.

1 Introduction

Many important decisions require sacrificing immediate reward in the pursuit of greater payoffs in the future. Incorporating potential future rewards requires solving a Markov decision process. The computational challenge of solving this problem may partly explain why people tend to underappreciate future rewards relative to immediate rewards (Myerson & Green, 1995).

Inspired by the use of *shaping rewards* in reinforcement learning (Ng, Harada, & Russell, 1999), Lieder and Griffiths (2016) proposed that this problem can be ameliorated by providing people with *pseudo-rewards* that align short term reward with long term value. They showed that optimal pseudo-rewards (i.e., the value of the next state minus the value of the current state) enable myopic agents (i.e., agents maximizing immediate rewards) to follow the optimal policy and improve human performance in sequential decision problems.

In the real world, however, it is generally infeasible to provide pseudo-rewards after every action a person takes. Furthermore, computing optimal pseudo-rewards exactly is intractable for the decision problems that people face in real-life. Instead, a decision support system may only have access to the approximate values of a limited number of states. Fortunately, although people may not be able to look far into the future, research on human planning has shown that they are not entirely myopic either (Morris & Ward, 2004). One strategy people use to approximate optimal planning is to set subgoals (Newell, Simon, et al., 1972), an approach that has also been applied in artificial intelligence (Shivashankar, Kuter, Nau, & Alford, 2012).

However, to use subgoals effectively one must choose good subgoals, and humans sometimes rely on simple heuristics and habits when setting subgoals (Cushman & Morris, 2015). Perhaps then, humans and decision support systems have complementary stregnths and weaknesses. Humans can often achieve subgoals, but they may not always select the best subgoals to pursue. Decision support systems, on the other hand, cannot feasibly advise on every decision, but they may have information about the values of certain states. This suggests that a decision support system can improve human decision making by helping people choose better subgoals using its knowledge of valuable states.

We propose that a decision support system can inform peoples' choice of subgoals by providing *sparse pseudo-rewards* to convey the value of potential subgoal states that are within reach. People could then select one of these states as their target by combining their knowledge of proximal rewards with the state values conveyed by pseudo-rewards. Here, we investigate the potential benefits and perils of this approach through mathematical analysis and behavioral experiments.

2 Optimal sparse pseudo-rewards for non-myopic agents

Lieder and Griffiths (2016) proved that pseudo-rewards generated by a potential function of the optimal value function lead a *myopic* agent (one that greedily maximizes immediate reward) to follow the globally optimal policy, taking $r'(s_t, a_t, s_{t+1}) = r(s_t, a_t, s_{t+1}) + f(s_t, a_t, s_{t+1})$ where $f(s_t, a_t, s_{t+1}) = V^\star(s_{t+1})\gamma - V^\star(s_t)$. Here, we extend this result, showing that sparse pseudo-rewards can provide the same benefit for an agent with some ability to plan ahead. Let an N-step planner be the policy

$$\pi^N(s_t) = \arg\max_{a_t} \max_{a_{t+1}\cdots a_{t+n-1}} \sum_{i=0}^{N-1} r'(s_{t+i}, a_{t+i}, s_{t+i+1})\gamma^i \tag{1}$$

Lieder and Griffiths (2016) note that an optimal planning agent will still perform optimally when given their pseudorewards because of the shaping theorem (Ng et al., 1999). However, because sparse pseudo-rewards are necessarily not potential-based, the shaping theorem cannot apply. Thus, we begin by proving directly the optimality of dense pseudo-rewards in order to build intuition for the sparse case.

Lemma 2.1. *The N-step planner π^N is equivalent to the optimal policy π^* when the pseudo-reward function is that of Lieder and Griffiths (2016).*

Proof. We begin by breaking up the modified reward into its pieces.

$$\sum_{i=0}^{N-1} r'(s_{t+i}, a_{t+i}, s_{t+i+1})\gamma^i = \sum_{i=0}^{N-1} r(s_{t+i}, a_{t+i}, s_{t+i+1})\gamma^i + \sum_{i=0}^{N-1} f(s_{t+i}, a_{t+i}, s_{t+i+1})\gamma^i \tag{2}$$

Expanding the second term, we find that the fs telescope.

$$\sum_{i=0}^{N-1} f(s_{t+i}, a_{t+i}, s_{t+i+1})\gamma^i = -V^\star(s_t) + V^\star(s_{t+1})\gamma^1 \cdots - V^\star(s_{t+N-1})\gamma^{N-1} + V^\star(s_{t+N})\gamma^N = V^\star(s_{t+N})\gamma^N - V^\star(s_t) \tag{3}$$

Thus the N-step planner chooses the first action in a series of actions that maximizes the sum of rewards plus the difference in value between the final state and the current state. The value of the current state is constant under the maximization, so we can remove it.

$$\pi^N(s_t) = \arg\max_{a_t} \max_{a_{t+1}\dots a_{t+n-1}} \sum_{i=0}^{N-1} r(s_{t+i}, a_{t+i}, s_{t+i+1})\gamma^i + V^\star(s_{t+N})\gamma^N \tag{4}$$

Starting from the optimal policy and working backwards, we expand the recursive definition of $V^\star$ $N-1$ times to attain

$$\pi^\star(s_t) = \arg\max_{a_t} r(s_t, a_t, s_{t+1}) + \cdots + \max_{a_{t+N-1}} r(s_{t+N-1}, a_{t+N-1}, s_{t+N})\gamma^{N-1} + V^\star(s_{t+N})\gamma^N = \pi^N(s_t) \tag{5}$$

■

Note that this proof can be easily extended to stochastic MDPs by replacing the functions f, r and $V^\star$ with the expected values of the corresponding distributions.

Given that it may not be feasible to provide pseudo-rewards after every action, we would like would like to construct an f that has this optimality property but is also sparse (i.e. $f(s_t, a_t, s_{t+1}) = 0$ for most t). Unfortunately, it is likely impossible to construct such an f without making assumptions about the agent and environment. As shown in Equation 4, the key contribution of the pseudo-rewards is in the term $V^\star(s_{t+N})\gamma^N$, which comes from the pseudo-reward $f(s_{t+N-1}, a_{t+N-1}, s_{t+N})$. That is, the proof relies critically on the pseudo-reward at the final transition of each path considered while planning. In general, this could be any transition in the MDP, thus we cannot safely set f to 0 for any transition, precluding sparsity in the general case.

Designing optimal sparse-pseudo rewards may be impossible without knowing what paths the agent could consider. However, in some cases, this information may be available. Assume we know the current state of the agent s_t as well as its planning depth N. This information defines a set S_{t+N} of all possible states the agent could be in after N steps. We can then define pseudo-rewards relative to the current time step as

$$f_t(s, a, s') = \begin{cases} V^\star(s') & \text{if } s' \in S_{t+N} \\ 0 & \text{otherwise} \end{cases}$$

Lemma 2.2. *The N-step planner π^N is equivalent to the optimal policy π^* when the pseudo-reward function is f_t and the environment in non-cyclic.*

Proof. Substituting f_t into Equation 3, we find $\sum_{i=0}^{N-1} f_t(s_{t+i}, a_{t+i}, s_{t+i+1})\gamma^i = V^\star(s_{t+N})\gamma^N$ which differs from the original case only in the constant that is removed in Equation 4. Thus, we can follow the same steps starting at Equation 4 to show that π^N with shaping function f_t is equivalent to π^*. Note that the first step of this proof requires that f_t be 0 for all states the agent visits except s_{t+N}. This assumption holds in non-cyclic environments. ■

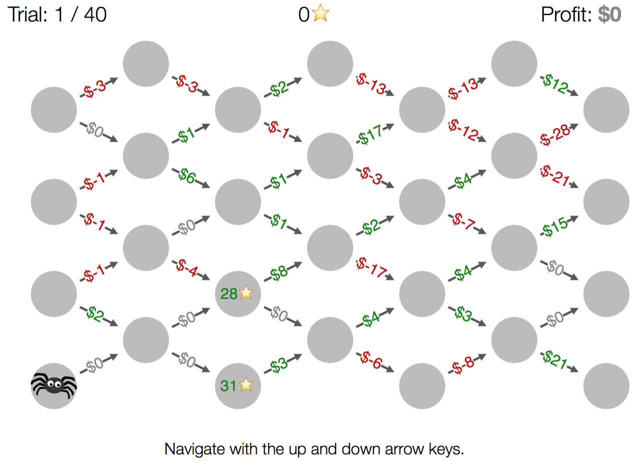
This formulation is less than ideal because it only applies to non-cyclic environments. Furthermore, the agent will never actually attain a pseudoreward because f_t moves a little further every step. Both problems can be solved by assuming that the N-step planner never plans past the nearest pseudo-reward (an assertion that we make without proof for lack of space). In this case we can reset f_t only when the agent attains the pseudo-reward. We follow this assumption in designing our experimental stimuli.

3 Testing the effects of sparse pseudo-rewards on planning

To investigate the effects of pseudo-rewards on planning, we employ a modified version of the paradigm of Lieder and Griffiths (2016) which presents a series of sequential decision problems with versus without pseudo-rewards. We hypothesized that spacing out the pseudo-rewards such that they occurred every N time steps would lead participants to form subgoals to reach states with high pseudo-rewards. This simplifies the planning problem, allowing participants to plan only as far as the next pseudo-reward, and thus making their planning process to be less error-prone and less time-consuming. We expected this benefit to be proportional to the reduction in planning distance. Thus, we predicted that participants would achieve higher scores and spend less time on trials with more frequent pseudo-rewards.

3.1 Methods

We recruited 88 participants on Amazon's Mechanical Turk using the psiTurk experimental framework (Gureckis et al., 2016). Participants were paid \$0.75 plus a performance-dependent bonus of up to \$1.85 for completing 40 trials



Navigate with the up and down arrow keys.

Figure 1: Screenshot of the experiment: Participants control the spider with the arrow keys to maximize profit. The number of stars conveys the value of a state.

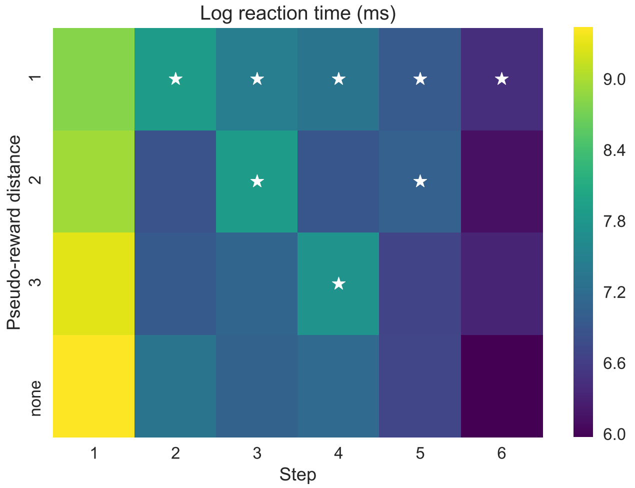


Figure 2: Planning time is higher on states with pseudo-rewards, indicated by stars.

(roughly 14 minutes of work). The experiment was programmed with the JsPsych library (De Leeuw, 2015) using the Mouselab-MDP plugin (Callaway, Lieder, Krueger, & Griffiths, in press). Trials were counterbalanced such that each reward structure occurred with each pseudo-reward distance an equal number of times across all participants.

A screenshot of the task is shown in Figure 1. Participants navigated a spider through the *Web of Cash*, aiming to attain the highest possible profit. (Their bonus was 5% of their profit on a random round). The graph was structured such that participants could only move rightwards, and only one row up or down on each step. This induces a long-term planning problem in which one may need to make a short-term suboptimal decision to reach a highly valuable state. To increase the importance of planning, rewards were drawn from a zero-centered normal distribution with variance increasing with depth, $\text{Normal}(0, 2 + 2d)$, and clipped to be integers. We selected stimuli that discriminated 1-step, 2-step, and 6-step planners by rejection sampling.

On some trials, pseudo-rewards, displayed as stars, were presented every 1, 2, or 3 steps. Only the pseudo-rewards at the nearest such column were visible; when one of those pseudo-rewards was attained, the next round appeared. Participants were explicitly informed that the number of stars on a state was the maximum total profit one could gain starting in that state. In MDP jargon, the number of stars on a state was equal to the value of that state. Thus a participant could take the optimal path by considering all paths up to the next column of pseudo-rewards and taking the path with the largest sum of rewards (dollars) and pseudo-rewards (stars).

3.2 Results

Planning time Before conducting our primary analyses, we sought evidence that the pseudo-rewards induced goal-targeted planning. We defined an intuitive behavioral signature of planning based on reaction times: If a participant chooses to plan ahead at a given state, her reaction time will be higher than average; however, once a plan has been formed, she can quickly execute the full sequence. Thus, if the pseudo-rewards induced subgoal-targeted planning, we would expect to see longer reaction times on states with pseudo-rewards because participants would need to re-plan after reaching the pseudo-reward induced subgoal.

Log reaction times broken down by step and pseudo-reward frequency are shown in Figure 2. We excluded the first action (the first column in Figure 2) from our analysis because one would expect unusually high reaction time at the first state regardless of the presence of pseudo-rewards. We constructed a mixed effects linear regression model over log reaction time at each step with one fixed effect for the presence of a pseudo-reward on that step and random effects for participant, stimulus, trial index, and step number. An Anova comparing the full model to the model with only random effects revealed a significant effect of pseudo-rewards on planning time ($\beta = 0.708$, $\chi^2(1) = 619.37$, $p < 0.001$), indicating that pseudo-rewards induced goal-targeted planning.

Score rate We predicted that pseudo-rewards would either increase the quality of actions, reduce the time spent planning, or both. To capture both of these possibilities in one metric, we defined *score rate* to be the average score per second. Results are plotted in Figure 3. For each measure (score, time, score rate), we constructed a mixed effects linear regression model with one fixed effect for distance between pseudo-rewards (or 6, the number of steps, when pseudo-rewards were

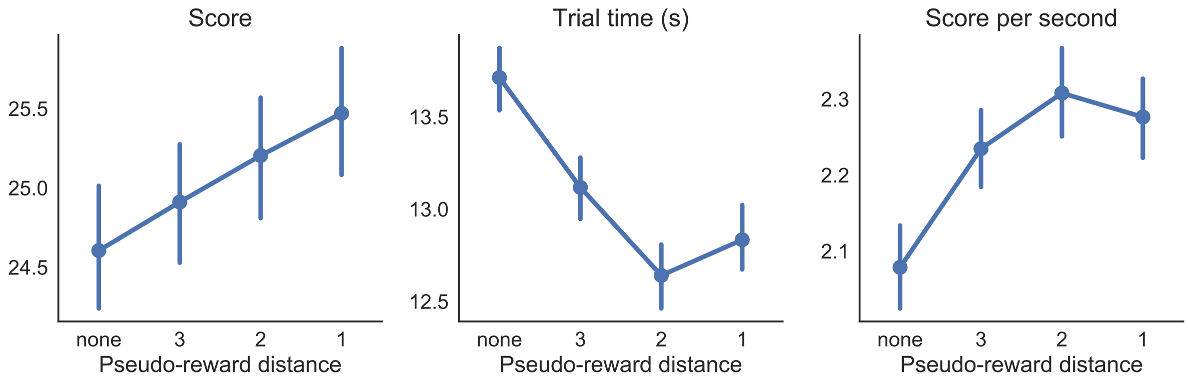


Figure 3: Average score, trial time, and score rate (score per second) for different frequencies of pseudo-rewards. Error bars show 68% confidence intervals by bootstrapping.

absent) and random effects for participant, stimulus, and trial index. The effect of distance between pseudo-rewards on score rate was statistically significant ($\beta = -0.045$, $\chi^2(1) = 15.14,\ p < 0.001$). The effect on score ($\beta = -0.166$, $\chi^2(1) = 4.90,\ p < 0.05$) and trial time were also significant ($\beta = 0.204$, $\chi^2(1) = 44.74,\ p < 0.001$).

Unwarranted sacrifices So far we have seen that sparse subgoal-directed planning induced by pseudo-rewards can have positive effects. However, there may be negative side effects. Increasing the salience of valuable states may lead participants to neglect the cost of reaching the state, or the many small rewards along a path to a less valuable state. We term a suboptimal plan made in tenacious pursuit of a goal an *unwarranted sacrifice*. This effect can be quantified in our paradigm as traveling to the state with maximal value at a given time step—perhaps highlighted by the largest pseudo-reward—when that state does not lie on the optimal path. Let $s_t^V = \arg\max_{s \in S_t} V^*(s)$ be the state with maximal value at step t. Similarly, let s_t^* be the state at step t of the optimal path. We can define a state s_t as the result (and perhaps also the cause) of an unwarranted sacrifice if $s_t = s_t^V \wedge s_t \neq s_t^*$. A χ^2 test revealed a significant effect of pseudo-rewards on unwarranted sacrifices ($\chi^2(1) = 12.52,\ p < 0.001;$), supporting the hypothesis that pseudo-rewards increase the likelihood that people will pursue a valuable state when doing so is suboptimal.

4 Conclusion

We found that pseudo-reward induced subgoals enabled people to solve sequential decision problems more efficiently, suggesting that pseudo-rewards could be used in real-world settings to help people make better decisions. However, our results also suggest that this approach is not without perils: When choosing subgoals, people may neglect the cost of achieving them. In future research, we will examine the extent to which pseudo-rewards exacerbate this effect, and what can be done to ameliorate it.

References

Callaway, F., Lieder, F., Krueger, P. M., & Griffiths, T. L. (in press). Mouselab-MDP: A new paradigm for tracing how people plan. In *The 3rd multidisciplinary conference on reinforcement learning and decision making.*

Cushman, F., & Morris, A. (2015). Habitual control of goal selection in humans. *Proceedings of the National Academy of Sciences*, *112*(45), 13817-13822. doi: 10.1073/pnas.1506367112

De Leeuw, J. R. (2015). jsPsych: A javascript library for creating behavioral experiments in a web browser. *Behavior Research Methods*, *47*(1), 1–12.

Gureckis, T. M., Martin, J., McDonnell, J., Rich, A. S., Markant, D., Coenen, A., . . . Chan, P. (2016). psiTurk: An open-source framework for conducting replicable behavioral experiments online. *Behavior research methods*, *48*(3), 829–842.

Lieder, F., & Griffiths, T. L. (2016). Helping people make better decisions using optimal gamification. In *Proceedings of the 38th annual meeting of the cognitive science society* (pp. 2075–80). Austin, TX: Cognitive Science Society.

Morris, R., & Ward, G. (2004). *The cognitive psychology of planning*. Psychology Press.

Myerson, J., & Green, L. (1995). Discounting of delayed rewards: Models of individual choice. *Journal of the experimental analysis of behavior*, *64*(3), 263–276.

Newell, A., Simon, H. A., et al. (1972). *Human problem solving* (Vol. 104) (No. 9). Prentice-Hall Englewood Cliffs, NJ.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml* (Vol. 99, pp. 278–287).

Shivashankar, V., Kuter, U., Nau, D., & Alford, R. (2012). A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proceedings of the 11th international conference on autonomous agents and multiagent systems-volume 2* (pp. 981–988).

Adversarially Robust Policy Learning through Active Construction of Physically-Plausible Perturbations

Ajay Mandlekar*
Department of Computer Science
Stanford University
Stanford, CA 94305

Yuke Zhu*
Department of Computer Science
Stanford University
Stanford, CA 94305

Animesh Garg*
Department of Computer Science
Stanford University
Stanford, CA 94305

Li Fei-Fei
Department of Computer Science
Stanford University
Stanford, CA 94305

Silvio Savarese
Department of Computer Science
Stanford University
Stanford, CA 94305

Abstract

Policy search methods in reinforcement learning have demonstrated success in scaling up to larger problem sizes beyond toy examples. However, deploying these methods on real robots remains challenging due to the large sample complexity required during learning and their vulnerability to malicious intervention. Model-based methods using simulated approximations of the target domains offer a possible solution, with the caveat that algorithms need to adapt across errors in modeling and adversarial perturbations. We introduce Adversarially Robust Policy Learning (ARPL), an algorithm that leverages active computation of physically-plausible adversarial examples during training to enable sample-efficient policy learning in the source domain and robust performance under both random and adversarial input perturbations. We also show that ARPL is analogous to improving the distance to uncontrollability for linear systems, which provides a guideline for a comparison of controller robustness in complex models. We evaluate ARPL on four continuous control tasks and show superior resilience to 18 different threat models across 100 policy instances for each task as compared to state-of-the-art robust policy learning methods. Code, data, and additional results are available at: https://stanfordrl.github.io/ARPL.

Keywords: Policy Learning, Adversarial Examples, Robust Control

Acknowledgements

This work is supported by the Toyota Center for AI Research at Stanford University.

**:equal contributions. All authors are affiliated to AI Lab, Stanford University, CA US; `{amandlek, yukez, animesh.garg, feifeili, ssilvio}@stanford.edu`

1 Introduction

Renewed research focus on policy learning methods in reinforcement learning have enabled autonomy in many problems considered difficult until recently, such as video games with visual input [9], the deterministic tree search problem in Go [12], robotic manipulation skills [7], and locomotion tasks [8].

As we move towards deploying learned controllers on physical systems around us, robust performance is not only a desired property but also a design requirement to ensure the safety of both users and system itself. Machine learning research has shown that a spectrum of models, including RL algorithms, are vulnerable to malicious attacks [1–3, 6]. Recent works have studied the existence of adversarial examples [4, 10, 13] and showed that such instances are not only easy to construct but are also effective against different models trained for the same task. While successful, policy search algorithms, such as variants of Q-Learning [9] and Policy Gradient methods [8, 11], are highly data intensive. Hence, a naïve approach that utilizes joint training over an ensemble of domains to achieve robustness can quickly become intractable.

This paper is a step towards addressing the problem of robustness to malicious attacks while maintaining data efficiency in training. The key intuition of this study is that – *adversarial examples can be used to actively choose perturbations during training in the source domain. This procedure exhibits robustness to both modeling errors and adversarial perturbations on the target domain without an increase in data-requirement.* In particular, we explore training in simulated continuous control tasks for evaluation across varied simulated target domains. We analyze the effect of perturbations on the performance via three kinds of modeling errors: model parameter uncertainty, process noise, and observation noise.

Summary of Contributions:

1. We demonstrate that deep RL policies are susceptible to both adversarial perturbations and model-mismatch errors.
2. We propose a method to synthesize physically-plausible adversarial perturbations for a white-box model. Further, we present Adversarially Robust Policy Learning method to use actively chosen adversarial perturbations for robust policy training.
3. We also relate the distance to uncontrollability in simple environments to control inputs, hence providing a potential justification for ARPL leading to improved robustness in more complex environments where analytic calculations are impractical.
4. We extensively evaluate our method in 4 simulated environments with many policy training variations in each and observe that training with ARPL results in improving cumulative reward.

2 Adversarially Robust Policy Learning

2.1 Adversarial Perturbations in Deep Neural Networks

Szegedy et al. [13] discovered the insightful fact that deep learning models are highly vulnerable to adversarial examples. Furthermore, these adversarial examples exhibit a remarkable generalization property - a variety of models with different parametrization are often fooled by the same adversarial example, even when these models are trained on different subsets of the training data.

Methods of creating adversarial examples rely upon maximizing the prediction error subject to a constraint on the perturbation size. In image classification, the objective is switching prediction classes, while minimizing the perceived image perturbation. In reinforcement learning, the objective is to misguide the policy to output incorrect actions, while minimizing the change in either the input state, the dynamics model, or the observation model. One of the most prevalent methods for generating adversarial examples is the Fast Gradient Sign Method (FGSM) by Goodfellow et al. [4]. FGSM offers computationally efficiency at the cost of a slight decrease in attack success rate. The FGSM method makes a linear approximation of a Deep Neural Network and maximizes the objective in a closed form.

FGSM focuses on adversarial perturbations of an image input where the change in each pixel is limited by ε. With a linear approximation of a DNN, $\hat{\pi}(s) = w^{\mathrm{T}}s$, the optimal FGSM perturbation is defined as, $\delta = \varepsilon\,\mathrm{sign}(w)$. Since we define δ to be the perturbation, the output of the network on the adversarial example $\hat{s}$ is $\hat{\pi}(\hat{s}) = w^{\mathrm{T}}s + w^{\mathrm{T}}\delta$. Now assuming that the network output $\pi(s;\theta)$ is instead a non-linear function parametrized by θ, then a linearization of the loss function around the current input provides the following perturbation

$$\delta = \varepsilon\,\mathrm{sign}\big(\nabla_s \eta(\pi_\theta(s))\big) \qquad \texttt{(FGSM)} \tag{1}$$

where η is a loss function over the policy π, which is parametrized by parameters θ. Moreover, we note that a key assumption in FGSM is that the attacker has complete access to the target neural network – such as its architecture, weights, and other hyperparameters. This is a *white-box* setting. In contrast, in a *black-box* setting, the attacker does not have access to the parameters of the target network but only the output. Previous works have studied the black-box setting where gradient computation can be done numerically, and the attacker has unlimited query access to the oracle. A common approach is to re-train a separate model for the same input and output space and leverage the transferability of the adversarial examples to attack the target policy. We restrict our analysis to a white-box setting because training the policy with numerical gradient estimates is no more efficient than computing the gradient on the original policy.

2.2 Physically Plausible Threat Model

Consider a physical dynamical system:

$$x_{t+1} = f(x_t, u_t; \mu) + \nu \qquad \texttt{(Dynamics)}, \qquad z_t = g(x_t) + \omega \qquad \texttt{(Observation)} \tag{2}$$

where the `Dynamics` equation updates the state x with control input u according to a function f, parametrized by model parameters μ, and process noise ν. The `Observation` model maps the current state x to the observed state z with the observation function g and observation noise ω.

We use the full gradient method instead of the popular FGSM. FGSM is primarily designed for images where the state is high dimensional and approx. IID. However, for dynamical models, the state space is structured with different domains; hence a fixed unit step size can result in scaling issues.

We use an isometrically scaled version of the full gradient

$$\delta = \varepsilon \, \nabla_s \eta(\pi_\theta(s)) \qquad \texttt{(ARPL)} \tag{3}$$

where η is a loss function over the policy π, which is parametrized by parameters θ.

Type of Perturbation: A malicious adversary can change either of the three quantities μ, ν, or ω. A change in μ can be equated to dynamics noise, i.e. uncertainty in physical parameters such as mass, friction, and inertia, while ν and ω correspond to direct perturbations of state and observation. Prior work in [2, 6] only examines perturbations to the current state in image space, i.e. ν, which is often not physically plausible. We perform perturbations to process noise ν by adding gradient based perturbation to the state of the system. Similarly, we add observation noise to ω by changing the observation while preserving the system state. Adversarial perturbation on dynamics noise through model parameters μ requires state augmentation $\bar{s} = [s, \mu]^T$, and only the latter component of the gradient is used $\nabla_{\bar{s}} = [0, \nabla_\mu]$.

We maintain physical plausibility of all perturbations through the projection of the perturbed state to its respective domain, i.e. the state space for s and bounded variation in $\mu \in [0.5\mu_0, 1.5\mu_0]$, where μ_0 is nominal (source) dynamics.

Modes of Perturbation: We build two threat modes: *Adversarial* and *Random*. For noise in states (ν) and observation (ω), adversarial states are calculated using δ, while random perturbations are uniformly sampled from $[-\delta, \delta]$. For dynamics noise, μ is set to be a uniform sample in $[0.5\mu_0, 1.5\mu_0]$ at each time iteration. For adversarial dynamics noise, we first get a random sample as before, then add a gradient δ evaluated at $\mu_{adv} \sim U(0.5\mu_0, 1.5\mu_0)$.

Frequency of Perturbation: The parameter $\phi \in [0, 1]$ determines the frequency of applying adversarial (or random) updates. At each time step, an update is applied with prob. $Bern(\phi)$. When $\phi = 0$, only the initial time step is perturbed in each episode.

The three perturbation types described above combined with two modes of perturbation and three levels of perturbation frequency result in a total of 18 threat models. Each model is also compared with the nominal source model with no changes as a baseline.

2.3 Robust Training with Adversarial Perturbations

Direct Policy Optimization methods utilize batch trajectory sampling for gradient estimates as in [11]. The core of the ARPL operates by modifying the trajectory rollouts to include trajectory perturbation. In the most general setting, at each iteration, a trajectory is rolled out, and an adversarial perturbation is added to the model with probability ϕ at each time step along the trajectory. The exact operation in `add_perturbation` depends on the choice of threat model. A gradient update to the policy parameters is then made after a rolling out a batch of k trajectories.

ARPL achieves robustness by adding adversarial model variation in each rollout. However, training on adversarial examples is different from other data augmentation schemes. In supervised models, the data is augmented with *a priori* transformations such as translation and rotation which are expected to occur in the test set. By contrast, adversarial perturbations rely on the online generation of scenarios that not only expose flaws in the ways that the model conceptualizes its decision function, but also are likely to occur naturally

3 Robustness as Distance to Uncontrollabilty

A dynamical system is said to be controllable if there exists a sequence of control inputs that can transfer the system to any desired state. Model perturbations can turn a controllable dynamical system into one that is uncontrollable. For a controllable linear system ($\dot{x} = Ax + Bu$), the distance to uncontrollability, d_{uc} can be characterized by the smallest singular value of the matrix $[A - \lambda I, B]$,

$$d_{uc} = \min_\lambda \sigma_{\min}\big([A - \lambda I, B]\big) \tag{4}$$

Consider an infinite-horizon time-invariant Linear Quadratic Regulator (with standard notation), the form of the optimal controller is $u = -Kx$. Here K is given by $K = R^{-1}(B^T P)$, and P is the solution of a Continuous time Algebraic Riccati Equation (CARE):

$$A^T P + A - (PB)R^{-1}(B^T P) + Q = 0$$

Proposition 1 *Further, we note that under the assumption that $(A;B)$ is controllable, $(A;Q)$ is observable, P is positive definite solution to CARE, with Q is positive semi-definite, R is positive definite, we have:*

$$d_{uc} \geq \frac{\lambda_{min}(Q)}{||P||\sqrt{1+\frac{\lambda_{min}(Q)}{\lambda_{min}(R)}})} \tag{5}$$

where $\lambda_{min}(\cdot)$ is smallest eigenvalue. We omit the proof and refer the reader to Theorem 2 of the analysis in [5], from which the above can be derived.

Proposition 1 implies that the distance to uncontrollability under perturbations to the matrices $(A;B)$ is inversely proportional to the matrix norm of P, and by extension the total control effort. Intuitively, if the control approximately saturated in order to achieve or maintain stability, then the system can be destabilized by small perturbations to the model, and vice-versa. While this result is hard to derive for non-linear systems, we can use this result as a guideline to compare the robustness of controllers under the same set of perturbations on the same task by directly comparing the control effort needed to achieve stability.

4 Experimental Evaluation

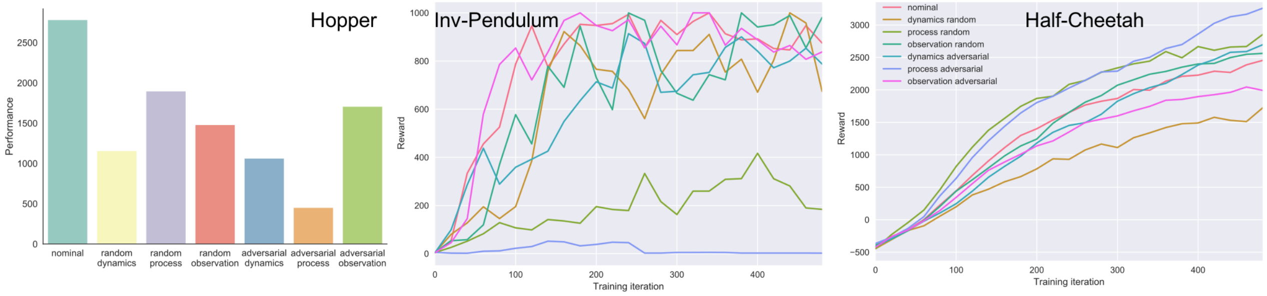


Figure 1: (a) The nominal `hopper` agent's performances against different types of random and adversarial perturbations in comparison with the nominal performance. (b) A comparison between the learning curve of the `half-cheetah` and the `inverted pendulum` agents.

Experimental Setup: We evaluated our proposed ARPL algorithm on four continuous control tasks – `inverted pendulum`, `half-cheetah`, `hopper`, and `walker-2d` using the MuJoCo physics simulator [14]. These tasks involve complex non-linear dynamics and direct torque control on the actuated joints. Under-actuation and discontinuous contact render these tasks a challenging benchmark for reinforcement learning methods. To understand our model's robustness with physical plausibility, we use a low-dimensional state representation that captures the joint angles and positions in these tasks. In such cases, perturbations on the state vectors naturally lead to a new state that is physically realizable in the environments.

We use the state-of-the-art trust region policy optimization (TRPO) method [11] to learn a stochastic policy using neural networks. The policy is parametrized by a Gaussian distribution, where its mean is predicted by a network with two hidden layers, 64 units per layer, and tanh as the non-linearity. The network also learns to output the standard deviation of the Gaussian policies. For the loss function η that ARPLuses to generate adversarial perturbations, we use $\eta(\mu_\theta) = \|\mu_\theta\|_2^2$, where μ_θ is the output of the mean network with parameters θ.

Existence of Physical Plausible Perturbations: We illustrate that a nominal policy (a policy trained without ARPL) has high sensitivity with respect to uncertainty from different sources. In particular, we examine perturbations as a result of dynamics noise, process noise, and observation noise introduced in Sec. 2.2. Fig. 1 illustrates the behavior of a nominal policy trained on the `hopper` environment in different setups. The left bar shows its performance in the nominal environment where the agent has been trained. We also show the agent's performance after applying random and adversarial perturbations to the environment. The most severe impact is caused by the adversarial perturbation on states (process noise) using the method described in Sec. 2.2. This illustrates the lack of robustness of the nominal policy regarding all three types of physical plausible perturbations.

Improving Model Robustness using Adversarial Training: Here we evaluate the effectiveness of our robust training method proposed in Sec. 2.3. This analysis is performed on all of the four environments on the three types of noises.

Fig. 1 shows the patterns of the agent's learning curves in seven different setups, evaluated in the same setup during training. In the robust training methods, we set the gradient multiplier factor ε to 0.1. Our robust training method exhibits comparable sample complexity with the nominal policy training. However, we observe distinctive behaviors in the `inverted pendulum` (left) and `half-cheetah` (right) environments - the `inverted pendulum` is much more sensitive to process noise. As a regulation task, a large perturbation on the states of the inverted pendulum can often lead to a large tilt that the agent cannot recover from. Consequently, the agent failed to learn a good policy when training with process perturbations. The best performance on the `inverted pendulum` task is achieved by training with adversarial dynamics noises. To better understand the robustness of the inverted pendulum agents, we quantitatively evaluate three agents that have been trained in 1) the nominal environment, 2) with random noise, and 3) and with adversarial perturbations. Fig. 2 shows the agents' performance under different combinations of dynamics configurations. Among the

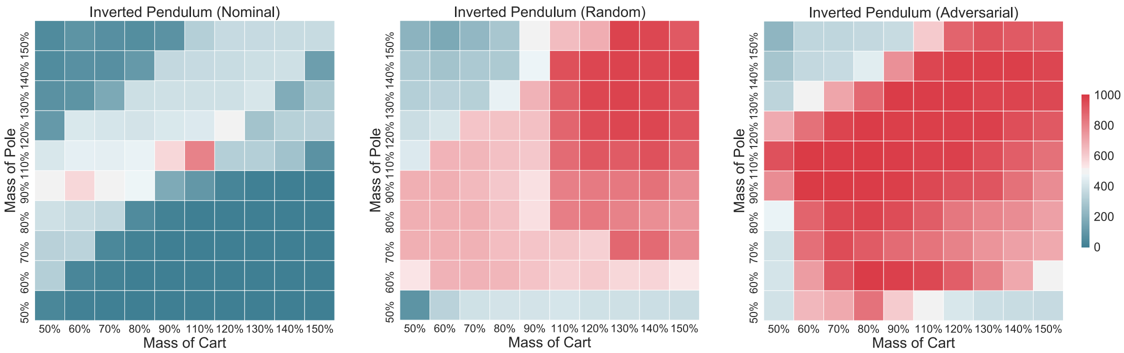


Figure 2: Policy robustness of the `inverted pendulum` agents with respect to varying dynamics configurations.

three setups, the agent trained with the adversarial dynamics is able to perform well in a broader set of the mass values of the body parts.

Robustness as Controllability: We also investigated the control effort exerted by the agents on the `inverted-pendulum` regulation task. We found that when testing the agents in the nominal domain, the three methods: adversarial (ARPL), random training, and nominal model result in a normalized RMS control effort of [0.15, 0.38, 1.75] (10-rollout mean). The higher control effort in nominal and random models confirms our hypothesis that these models are more susceptible to instability, as stated in Section 3.

5 Discussion and Future Work

Adversarial examples have been widely studied since they were first introduced by Szgedey et al. [13], especially in the context of classification. Recent work extended the idea to reinforcement learning [2, 6], showing that similar adversarial attack techniques can be directly applied to a variety of standard deep RL models. Some countermeasures attempt to incorporate perturbed adversarial examples to retrain the network [4], but one limitation of adding small perturbations for generating adversarial images is that it might not preserve the statistics of real-world data, so these adversarial examples might never exist in the physical world. Our work focuses on continuous control tasks, where the states naturally correspond to joint angles and positions. Therefore, we can use the perturbation method to generate a physically-plausible adversarial state, and perform formal analysis on physical systems. Future work involves exploring the techniques in high-dimensional sensory data, as well as methods for black-box adversarial attacks and their countermeasures.

References

[1] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, “Can machine learning be secure?”, in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, ACM, 2006, pp. 16–25 (cit. on p. 1).

[2] V. Behzadan and A. Munir, “Vulnerability of deep reinforcement learning to policy induction attacks”, Jan. 16, 2017. arXiv: `1701.04143v1 [cs.LG]` (cit. on pp. 1, 2, 4).

[3] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, “Evasion attacks against machine learning at test time”, in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2013, pp. 387–402 (cit. on p. 1).

[4] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples”, Dec. 20, 2014. arXiv: `1412.6572v3 [stat.ML]` (cit. on pp. 1, 4).

[5] C. He, “On the distance to uncontrollability and the distance to instability and their relation to some condition numbers in control”, *Numerische Mathematik*, vol. 76, no. 4, 1997 (cit. on p. 3).

[6] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, “Adversarial attacks on neural network policies”, Feb. 8, 2017. arXiv: `1702.02284v1 [cs.LG]` (cit. on pp. 1, 2, 4).

[7] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection”, *ArXiv preprint arXiv:1603.02199*, 2016 (cit. on p. 1).

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning”, *ArXiv preprint arXiv:1509.02971*, 2015 (cit. on p. 1).

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning”, *Nature*, vol. 518, no. 7540, pp. 529–533, 2015 (cit. on p. 1).

[10] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples”, Feb. 8, 2016. arXiv: `1602.02697v3 [cs.CR]` (cit. on p. 1).

[11] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, “Trust region policy optimization”, *ICML*, 2015 (cit. on pp. 1–3).

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of go with deep neural networks and tree search”, *Nature*, vol. 529, no. 7587, pp. 484–489, 2016 (cit. on p. 1).

[13] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks”, Dec. 21, 2013. arXiv: `1312.6199v4 [cs.CV]` (cit. on pp. 1, 4).

[14] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control”, in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, IEEE, 2012, pp. 5026–5033 (cit. on p. 3).