Leveraging Off-Policy Prediction in Recurrent Networks for Reinforcement Learning

by

Matthew Schlegel

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

Partial observability—when the senses lack enough detail to make an optimal decision—is the reality of any decision making agent acting in the real world. While an agent could be made to make due with its available senses, taking advantage of the history of senses can provide more context and enable the agent to make better decisions. This thesis investigates recurrent architectures to learn agent state (a summarization of the agent's history), and identifies some modifications—inspired by predictive representations of state—to enable efficient learning in (continual) reinforcement learning. First, I contribute to standard recurrent neural networks trained through back-propagation through time. This contribution provides pragmatic recommendations for incorporating action information into a recurrent architecture, and through extensive empirical investigations shows the trade-offs of several techniques. Second, I develop a recurrent predictive architecture which uses temporal abstractions—predictions in the form of general value functions—as the basis for its state representation. I show advantages of this architecture over standard recurrent networks in a continuing reinforcement learning domain, derive an objective and corresponding learning algorithm, and discuss several added concerns when using this architecture—such as discovery, what types of networks can be constructed, and off-policy prediction.

Preface

The following thesis contains work published in various workshops, conferences, and journals. Specifically, Chapter 3, 4, 5, 6 and 8 are based on journal publications (Schlegel, Jacobsen, et al. 2021; Schlegel, Tkachuk, et al. 2022). Chapter 7 was presented at the Reinforcement Learning and Decision Making workshop (Schlegel and M. White 2022). Chapter 9 is based on work presented in conference proceedings (Schlegel, W. Chung, et al. 2019).

> Matthew Schlegel, Wesley Chung, Daniel Graves, Jian Qian, and Martha White (2019). "Importance resampling for off-policy prediction." In: *Advances in Neural Information Processing Systems* 32
>
> Matthew Schlegel, Andrew Jacobsen, Zaheer Abbas, Andrew Patterson, Adam White, and Martha White (2021). "General value function networks." In: *Journal of Artificial Intelligence Research*
>
> Matthew Schlegel and Martha White (2022). "Predictions Predicting Predictions." In: *Multidisciplinary Conference on Reinforcement Learning and Decision Making*
>
> Matthew Schlegel, Volodymyr Tkachuk, Adam M White, and Martha White (2022). "Investigating Action Encodings in Recurrent Neural Networks in Reinforcement Learning." In: *Transactions on Machine Learning Research*

While the focus of the thesis is on a core set of publications during my PhD, I contributed to a number of publications. While these papers influenced my thinking during the development of the central projects in this thesis, they are (mostly) unrelated to the topics discussed below.

Raksha Kumaraswamy, Matthew Schlegel, Adam White, and Martha White (2018). "Context-dependent upper-confidence bounds for directed exploration." In: *Advances in Neural Information Processing Systems* 32

Andrew Jacobsen, Matthew Schlegel, Cameron Linke, Thomas Degris, Adam White, and Martha White (2019). "Meta-descent for online, continual prediction." In: *Proceedings of the AAAI Conference on Artificial Intelligence.* Vol. 33. 01, pp. 3943–3950

Dhawal Gupta, Gabor Mihucz, Matthew Schlegel, James Kostas, Philip S Thomas, and Martha White (2021). "Structural Credit Assignment in Neural Networks using Reinforcement Learning." In: *Advances in Neural Information Processing Systems* 34

Matthew McLeod, Chunlok Lo, Matthew Schlegel, Andrew Jacobsen, Raksha Kumaraswamy, Martha White, and Adam White (2021). "Continual Auxiliary Task Learning." In: *Advances in Neural Information Processing Systems* 34

To future generations,

may our work benefit the many.

The first principle is that you must not fool yourself—and you are the easiest person to fool. So you have to be very careful about that. After you've not fooled yourself, it's easy not to fool other scientists. You just have to be honest in a conventional way after that.

– Richard Feynman, **Cargo Cult Science**, 1998

Acknowledgements

I have been very fortunate to have met and worked with many brilliant and encouraging mentors, friends, and colleagues. I wish I could thank all those who made the journey to get a Ph.D. possible and (mostly) enjoyable, but then this section would be longer than the following thesis.

First, my two co-supervisors Martha White and Adam White are more than I could have asked for as mentors and friends. Meeting them at Indiana University - Bloomington will always be one of the most impactful moments in my life. I will always be grateful to have been part of their labs and could not see myself finishing my degree without their support and encouragement. I have been amazed at the passion and drive they put into their research and hope to emulate them as I progress in my career. Along with my supervisors, I am deeply grateful to my committee—Michael Bowling, Matthew Guzdial, and André Barreto— for engaging with my research and providing valuable feedback. Finally, I want to thank all the collaborators of the work presented in this thesis: Andrew Jacobsen, Andy Patterson, Zaheer Abbas, Wesley Chung, Daniel Graves, Jian Qian, and Volodymyr Tkachuk. Your contributions have made this thesis significantly better. I look forward to working together in the future and continuing to be colleagues and friends.

I am deeply grateful for the support and friendship of my fellow students and colleagues. Special appreciation goes to Alex Kearney for the continued support, friendship, and smart-witty banter that made the hard times fly by. There is always time for (gluten-free) pasta, especially at 1 a.m. in Toronto. I also want to thank Andrew Jacobsen for his consistent stream of challenging music suggestions, his eccentric and exciting behavior at drinks, and his friendship through the years. I want to thank Shibhansh Dohare for getting me deeply embedded in the climbing world and making exercise in the last few years of my Ph.D. enjoyable. Also, thank you to Eric Graves for being a great introduction to the department and for having long discussions about tea, life, and research as we made tea for TTTs. I want to acknowledge Marlos Machado and Craig Sherston for being role models as senior Ph.D. students. I especially want to thank Marlos for the early morning conversations and advice in the first year of my degree. I also want to acknowledge Johannes Günther for his friendship and mentorship in doing research and finding my voice as a person and researcher. Many thanks also to Andy Patterson and Cam Linke (along with AJ) for the many fun conversations and drinks throughout the years, Roshan Shariff for his wisdom and engaging discourse on anything in mathematics, Raksha Kumaraswamy for being the best desk neighbor both here and at Indiana University, and Yangchen Pan for his mentorship and encouragement in my first research project. I want to send my appreciation and thanks to J. Fernando Hernandez Garcia, Abhishek Naik, Niko Yasui, Dylan Brenneis, Andy Wong, Brian Tanner, James Bell, Khurram Javed, Banafsheh Rafiee, Kirby Banman, Alex Lewandowski, and Chunlok Lo.

Finally, I want to thank my family and friends (those outside the department) for supporting me through these tumultuous years. My Mom and Dad for being my biggest supporters and advocates; Sarah Nagelvoort for being not only a sister but a friend, mentor, and listening post; Mark Nagelvoort for being the best brother-in-law ever; Henry and Evelyn Nagelvoort for letting me play with their toys when I'm in Bloomington; Roshni Sajiv Kumar for her support and partnership, I am grateful for your compassion and patience as I was preparing for my defense; Dan Kenneally, Pat Miller, and Ed Cowie for being great friends for an uncountable number of years and always being available to have wild conversations about AI's takeover of the world; and finally, Jonathan Song for his help in a difficult time.

My work was made possible by the generous support of IVADO, and the Alberta Machine Intelligence Institute (Amii). The computing resources used for this research were provided by Compute Canada (now Digital Research Alliance of Canada).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

One goal of machine intelligence research is to design and understand an agent which interacts with its world through a continuous stream of interactions, consistently getting better through feedback. Continual reinforcement learning (Ring 1994; Khetarpal et al. 2022) is one paradigm used to build agents which continually improve. Similar to reinforcement learning (RL), a continual RL agent seeks to maximize the cumulative sum of rewards—which provides a simple yet effective path towards learning behavior. The main attribute of continual RL is that there is an unending stream of sensorimotor interactions that the agent must sift through online. This is different from many instantiations of reinforcement learning where the agent often gets to the goal or the end of a maze and then is sent back to the beginning to continue learning—also known as episodic reinforcement learning—in continual RL the agent never reaches a point where it is reset to a constrained starting distribution. The agent must also balance computation to make decisions in real-time, while also learning from recent observations. Finally, the agent must handle a world which is constantly changing over time. These properties make the continual RL problem challenging, but applicable to a wide range of real-world applications.

Continual RL systems in the real-world will inevitably have to make decisions when the information available is insufficient to make an optimal choice (also known as partial observability). For example, a rolling robot which observes its world through a forward facing camera or other agent-centric (centering on the agent) sensors. The agent can view what is in its immediate senses, say a ball in-front of the camera. If the agent rotates the ball can disappear off the edge of the agent's vision, but the ball is still there physically. Or if the ball were to be kicked to the other side of a wall, the ball still exists but is visually obscured by the wall. These environment attributes could result in conditions in which the agent could never predict the future given its senses. In situations like these, where the agent has insufficient context, the world appears non-stationary: the world responds differently when the agent acts according to the same set of senses.

In the face of partial observability, the main approach developed in machine learning is to summarize the past to provide the agent context which is missing from its current senses. In the rolling robot example, the agent can keep a history to judge how long the ball has been out of the agent's immediate vision observations. The agent could use this information to move towards the ball while actively observing another agent who may steal the ball, or know to go over the wall to retrieve the ball and continue playing. In the past, artificial intelligence research focused on outlining all possible scenarios the agent could face (Durkin 1996). These systems faced challenges in that humans were notoriously poor at outlining why they made a decision in a given scenario. A raw history of observations has also been used to some effect in the past (McCallum 1993; McCallum et al. 1996), but these face many challenges in developing a long enough memory. The goal of representation learning—which is embodied by the current wave of deep learning—is to construct compact memories of the past in order to aid decision making.

In RL, the most widely used approaches for representation learning are based on ideas developed for supervised learning (SL). Recurrent neural networks (RNNs) have been established as an important tool for learning predictions of data with temporal dependencies. They have been primarily used in language and video prediction (Mikolov et al. 2010; Oh et al. 2015; T. Wang and Cho 2016; Saon et al. 2017; Y. Wang et al. 2018), but have also been used in traditional time-series forecasting (Bianchi et al. 2017). Recurrent networks have also been applied to the RL problem (Onat et al. 1998; Bakker 2002; Wierstra et al. 2007; Hausknecht and Stone 2015; Heess et al. 2015; Igl et al. 2018; Zhu et al. 2018), but often these applications focus on how to fit the RL problem into what will work with the standard recurrent training approaches. Recently, transformers (Vaswani et al. 2017) have become a widely used alternative to recurrent architectures in natural language processing. Transformers have also shown some success in reinforcement learning but either require the full sequence of observations at inference and learning time (Mishra et al. 2018; Parisotto et al. 2020) or turn the RL problem into a supervised problem using the full return as the training signal (Chen et al. 2021). This is again similar to the application of recurrent networks where the problem is modified to fit the architecture.

While we can—and arguably should—use the intuitions built in the SL setting for continual RL problems, these intuitions must be tested and re-justified as an agent if there is no-longer a well defined dataset with constrained histories and the agent must make decisions which effect the data stream, like in continual reinforcement learning. In this thesis, we ask

> How can ideas and algorithms from off-policy prediction and predictive state representations be used to make reinforcement learning methods more performant in partially observable, continual decision making tasks?

I use (continual) reinforcement learning to study recurrent architectures and provide novel observations and algorithms to learn representations in the RL setting. In the following sections, I outline the main contributions of this dissertation.

1.1 Incorporating Actions in Recurrent Networks

The contribution presented in Chapter 3 empirically evaluates architectural choices for encoding action in the update function of recurrent networks. The evaluation provides a simple, general recommendation to incorporate action information into recurrent neural networks through a multiplicative operation which improves prediction and control in reinforcement learning. Evidence for the preferred strategy is provided through an in-depth analysis of the prediction setting in a small example domain and in a variety of control problems.

1.2 A Novel Predictive Approach to Learning in the Face of Partial Observability

I develop a novel approach to incorporate ideas from predictive state representations—an approach to representation learning that is designed for temporal partially observable prediction problems—and recurrent neural networks, called a general value function network (GVFN). This contribution directly addresses the question posed above by directly restricting the state of a recurrent network to be learned through off-policy prediction. The approach is shown to be competitive to RNNs in several continual learning and time-series domains, often without needing history when learning. This contribution is split into three chapters.

- Chapter 4 introduces the architecture and relates it to recurrent neural networks and previous predictive approaches.
- Chapter 5 derives several learning algorithms for the architecture derived from an extension of the *Mean-Squared Projected Bellman Network Error*.
- Chapter 6 empirically compares the new architecture to standard recurrent neural networks in several continual learning prediction problems.

1.3 Designing and Generating Predictive Questions

The contribution presented in Chapters 7 and 8 investigates the set of predictive questions available to GVFNs and how these can be generated for the architecture discussed above. The impact of this contribution brings clarity to the set of predictive questions applicable to the GVFN architecture (Chapter 7) and progresses towards automatic deployment to a general set of partially observable settings (Chapter 8). These chapters use a collection of lemmas and empirical evidence to draw conclusions about the set of general value functions.

1.4 Off-policy Prediction using Importance Resampling

GVFNs require learning value functions off-policy. Off-policy prediction means learning the answer to a predictive question (i.e. GVF) using experience generated through a separate behavior policy—unrelated to the target policy of the GVF. Because a GVFN requires GVFs to be stable during the learning process, previous off-policy prediction algorithms—i.e. using importance sampling ratios—can cause significant problems if the behavior is very different from the target policy. This has an impact not only in learning GVFNs, but generally applying off-policy prediction to the deep learning setting.

The contribution presented in Chapter 9 defines a new off-policy prediction algorithm using importance resampling. The resulting estimator is shown to be more sample efficient than importance sampling, and robust to settings with large importance sampling ratios, while still being consistent. This estimator is widely applicable to reinforcement learning when using a replay buffer, but also is consequential for the predictive approaches developed in this thesis. The evidence provided is through several empirical experiments and theorems with proofs provided in the appendix.

Chapter 2

Background

In this thesis, I take the perspective that an agent is situated inside its environment and observes its world continually (Ring 1994; Ring 1997; Sutton, Modayil, et al. 2011). In this chapter, I provide the relevant general background. This includes background on reinforcement learning (RL) (including off-policy prediction and control), and learning under the constraint of partial observability. Specific background details related to certain solution methods will be presented closer to their relevant topics.

2.1 Reinforcement Learning

The problem setting considered in this thesis is (continual) reinforcement learning (RL). In short, a reinforcement learning agent seeks to maximize the accumulation of a reward signal by acting in the world. In this thesis, I am concerned with two learning problems in reinforcement learning. Specifically, I focus on the model-free prediction and control problem, but each share the same general framework. The agent-environment interaction consists of a stream of data (from the agent's senses), coming in at a consistent rate into the agent's central control systems. In most reinforcement learning, the agent-environment boundary is placed inside the agent's nervous system where parts of the agent's body which are defined through evolution are external to the learning process, and those that are learned and modified through an agent's lifetime are a part of the learning process. This enables RL researchers to focus on the core problem of learning a policy to maximize reward. Figure 2.1 depicts the

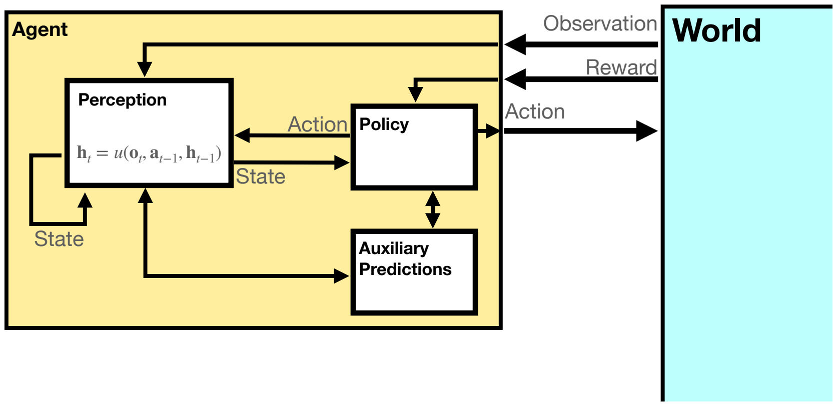


Figure 2.1: Diagram of the agent-environment interaction represented in reinforcement learning. The internals of the agent are representative of the agents discussed through this thesis. The perception box denotes the function used to construct agent state. The auxiliary prediction box is used in several ways throughout the thesis. Finally, the policy can either be learned or fixed depending on the experimental question being asked.

agent-environment interaction loop in RL.

In the agent's lifetime it observes its surroundings, takes actions, and receives rewards as the infinite sequence $\mathbf{o}_1, a_1, r_2, \mathbf{o}_2, \ldots, \mathbf{o}_t, a_t, r_{t+1}, \mathbf{o}_{t+1}, \ldots$. The observation $\mathbf{o}_t$ is the agent's window into the world through various sensing parts of its body. These can include a camera for vision, microphone for audio, lidar to measure distance from other objects, and many other analog-to-digital conversion technologies. The agent then selects an action A_t which is passed to the agent's actuators or sub-level control system. By performing this action, the agent receives a reward r_{t+1} and another observation $\mathbf{o}_{t+1}$ determined by the dynamics of the environment.

The agent-environment interaction can be formalized as a Markov decision processes (MDP). The underlying dynamics are defined by a tuple $(\boldsymbol{\Psi}, \mathcal{A}, \mathbf{P}, \mathcal{R})$. At time-step t given the state of the environment $\psi_t \in \boldsymbol{\Psi}$ [1] and action taken in that state $a_t \in \mathcal{A}$ the environment transitions to a new state $\psi_{t+1} \in \boldsymbol{\Psi}$ according to the state transition probability matrix $\mathbf{P} : \boldsymbol{\Psi} \times \mathcal{A} \times \boldsymbol{\Psi} \rightarrow [0, \infty)$ with a reward given by $\mathcal{R} : \boldsymbol{\Psi} \times \mathcal{A} \rightarrow \mathbb{R}$. In the fully observable case, the agent

[1]Typically, the environment state is denoted with s. Here I use ψ to more clearly distinguish it from the state of the agent 2.4.

receives the state of the environment as its observations $\mathbf{o}_t = \psi_t$, or an equivalent state which has the Markov property $p(\mathbf{o}_{t+1}|\mathbf{o}_t, a_t) = p(\mathbf{o}_{t+1}|\mathbf{o}_0, \ldots, \mathbf{o}_t, a_t)$. We discuss the partially observable case in Section 2.4. This thesis primarily assumes the discrete action setting, where the set of actions is a finite discrete set of values $\mathcal{A} \overset{\text{def}}{=} \{a_1, a_2, \ldots, a_n\}$.

The agent has several canonical internal components. A **policy** is a distribution over the space of actions $\pi : \boldsymbol{\Psi} \times \mathcal{A} \to [0, 1]$, which defines a way of interacting with the environment. Most often, a policy is notated as a conditional probability distribution where the probability of selecting an action given a state is $\pi(a|\psi)$. The return $G_t = \sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i}$[2] is the cumulative (discounted) reward into the future. A **value function** is a prediction of the future return the agent will obtain by following a policy. Specifically,

$$V^\pi(\psi) = \mathbb{E}_\pi[G_t|\Psi_t = \psi, A_t \sim \pi(\cdot|\Psi_t)] \tag{2.1}$$

with a state-action value function defined similarly

$$\mathcal{Q}^\pi(\psi, a) = \mathbb{E}_\pi[G_t|\Psi_t = \psi, A_t = a].$$

The operator $\mathbb{E}_\pi$ indicates an expectation with actions selected according to policy π. This thesis uses both state value functions and state-action value functions to do prediction and control. In the following sections I will 1) go into the specifics of the prediction (Section 2.2) and control (Section 2.3) problems as they relate to this thesis, and 2) introduce this framework to the partially observable case.

2.2 Prediction

The prediction problem in RL is that of learning value functions efficiently and accurately. This process can be used to improve an agent's policy through value iteration or policy iteration (Sutton and Barto 2018), or to learn temporal abstractions of the sensorimotor stream through options or general value functions (see Section 2.6 for more details). A value function can be learned

[2]Throughout this document, unbolded uppercase variables are random variables; lowercase variables are instances of that random variable; and bolded variables are vectors

either on-policy or off-policy through temporal difference learning. In this section, I introduce the on and off-policy prediction problem as used throughout this text.

As introduced above in Equation (2.1), a **value function** is a prediction of the future cumulative (discounted) reward received by following a policy π. GVFs encompass standard value functions, where the cumulant is a reward. Otherwise, GVFs enable predictions about discounted sums of others signals, when following a target policy π. These values are typically estimated using parametric function approximation, with weights $\mathbf{W} \in \mathbb{R}^d$ defining approximate values $V(\psi; \mathbf{W})$.

The simplest algorithm to learn the value function is through Monte-Carlo sampling. The brief of the algorithm is to get samples of the return starting in state Ψ following policy π, which are then averaged to receive the expected return. You can use the trajectories to estimate the returns for either first-visit to a specific state or on every visit, see (Singh and Sutton 1996; Sutton and Barto 2018) for more details. This algorithm only requires the environment to be episodic (i.e. clear terminations) and converges to the true value function as the number of rollouts grow.

Another approach to learning value functions is to take advantage of the Bellman equation through dynamic programming. The Bellman equation for the value function $V_\pi(\Psi)$

$$\begin{aligned}
V^\pi(\psi) &= \mathbb{E}_\pi[G_t | \Psi_t = \psi, A_t \sim \pi(\cdot|\Psi)] \\
&= \mathbb{E}_\pi[R_t + \gamma G_{t+1} | \Psi_t = \psi, A_t \sim \pi(\cdot|\Psi)] \\
&= \overline{R}(\psi, \pi(\psi)) + \gamma \sum_{\psi'} P(\psi' | \psi, A_t \sim \pi(\psi)) V^\pi(\psi')
\end{aligned}$$

where $\overline{R}(\psi, \pi(\psi))$ is the expected one-step reward for policy π in state ψ. The algorithm uses the transition dynamics of the environment $\mathbf{P}$ to iteratively calculate the value function through dynamic programming (Sutton and Barto 2018).

Temporal-difference learning combines advantages of both these algorithms, eliminating the need for environment dynamics (as in dynamic programming) and episodic environments (as in Monte-Carlo sampling). For tabular settings, TD learning follows the update rule

$$V(\Psi_t) \leftarrow V(\Psi_t) + \alpha \left[R_t + \gamma V(\Psi_{t+1}) - V(\Psi_t)\right].$$

The target for the temporal-difference learning algorithm is known as the TD target $R_t + \gamma V(\Psi_{t+1})$. TD bootstraps using the previous estimate of the return on the next state $V(\Psi_{t+1})$ (like dynamic programming) while sampling transitions from the environment following π (like Monte-Carlo sampling).

When using function approximation, the preferred approach is to follow the gradient taken of the value function with respect to the parameters of your function. This is known as the semi-gradient TD learning algorithm

$$\begin{aligned}\delta_t &= R_t + \gamma V(\Psi_{t+1}; \mathbf{W}_t) - V(\Psi_t; \mathbf{W}_t)\\ \mathbf{W}_{t+1} &\leftarrow \mathbf{W}_t + \alpha \delta_t \nabla_{\mathbf{W}} V(\Psi_t; \mathbf{W}_t).\end{aligned}$$

This update can be seen as minimizing the mean squared TD objective $\mathcal{L}(\Psi_t, \Psi_{t+1}, R_t) = (V(\Psi_t; \mathbf{W}_t) - U_t)^2$ assuming the bootstrapped target $U_t = R_t + \gamma V(\Psi_{t+1}; \mathbf{W}_t)$ has gradient $\nabla_{\mathbf{W}} U_t = 0$.

2.2.1 Off-policy Prediction

In off-policy prediction, transitions are sampled according to behavior policy, rather than the target policy. To get an unbiased sample of an update to the weights, the action probabilities need to be adjusted. Consider on-policy temporal difference (TD) learning, with update $\alpha_t \delta_t \nabla_{\mathbf{W}} V(\psi; \mathbf{W}_t)$ for a given $\Psi_t = \psi$, for learning rate $\alpha_t \in \mathbb{R}^+$ and TD-error $\delta_t \overset{\text{def}}{=} R_{t+1} + \gamma_{t+1} V(\Psi_{t+1}; \mathbf{W}_t) - V(\psi; \mathbf{W}_t)$. If actions are instead sampled according to a behavior policy $\mu : \mathbf{\Psi} \times \mathcal{A} \rightarrow [0, 1]$, then importance sampling (IS) is used to modify the update, giving the off-policy TD update $\alpha_t \rho_t \delta_t \nabla_{\mathbf{W}} V(\psi; \mathbf{W}_t)$ for IS ratio $\rho_t \overset{\text{def}}{=} \frac{\pi(A_t|\Psi_t)}{\mu(A_t|\Psi_t)}$. Given state $\Psi_t = \psi$, if $\mu(A|\psi) > 0$ when $\pi(A|\psi) > 0$, then the expected value of these two updates are equal. To see why, notice that

$$\mathbb{E}_\mu \left[\alpha_t \rho_t \delta_t \nabla_{\mathbf{W}} V(\Psi_t; \mathbf{W}_t) | \Psi_t = \psi\right] = \alpha_t \nabla_{\mathbf{W}} V(\psi; \mathbf{W}_t) \mathbb{E}_\mu \left[\rho_t \delta_t | \Psi_t = \psi\right]$$

which equals $\mathbb{E}_\pi \left[\alpha_t \rho_t \delta_t \nabla_{\mathbf{W}} V(\Psi_t; \mathbf{W}_t) | \Psi_t = \psi\right]$ because

$$\begin{aligned}\mathbb{E}_\mu \left[\rho_t \delta_t | \Psi_t = \psi\right] &= \sum_{a \in \mathcal{A}} \mu(a|\psi) \frac{\pi(a|\psi)}{\mu(a|\psi)} \mathbb{E}\left[\delta_t | \Psi_t = \psi, A_t = a\right] \\ &= \mathbb{E}_\pi \left[\delta_t | \Psi_t = \psi\right].\end{aligned}$$

Though unbiased, IS can be high-variance. A lower variance alternative is Weighted IS (WIS). For a batch consisting of transitions $\{(\psi_i, a_i, \psi_{i+1}, r_{i+1}, \rho_i)\}_{i=1}^n$, batch WIS uses a normalized estimate for the update. For example, an offline batch WIS TD algorithm, denoted WIS-Optimal below, would use update $\alpha_t \frac{\rho_t}{\sum_{i=1}^n \rho_i} \delta_t \nabla_{\mathbf{W}} V(\psi; \mathbf{W}_t)$. Obtaining an efficient WIS update is not straightforward, however, when learning online and has resulted in algorithms in the SGD setting (i.e. $n = 1$) specialized to tabular (Precup, Sutton, and Dasgupta 2001) and linear functions (Mahmood, van Hasselt, et al. 2014; Mahmood and Sutton 2015).

While the above objectives have been shown to work in a wide range of problem settings, there are a series of known counter examples where these algorithms do not converge. This is due to what is known as the deadly-triad in off-policy semi-gradient TD: off-policy updates, function approximation, and bootstrapping (Baird 1995; Sutton and Barto 2018; van Hasselt, Doron, et al. 2018). Removing any of these properties results in a convergent learning rule. Instead, minimizing an objective known as the *mean squared projected Bellman error* (MSPBE) converges to the TD fixed point. This objective minimizes the full Bellman error through a projection operator (Maei, Szepesvári, Bhatnagar, Silver, et al. 2009; Sutton, Maei, et al. 2009). Minimizing this objective results in several algorithms including one known as temporal-difference with corrections (TDC). For linear function approximation $V(\Psi_t; \mathbf{W}_t) = \mathbf{W}_t^\top \boldsymbol{\phi}_t$ (where $\boldsymbol{\phi}_t$ is the feature vector corresponding to state Ψ_t)

$$\begin{aligned}\mathbf{W}_{t+1} &\leftarrow \mathbf{W}_t + \alpha \delta_t \boldsymbol{\phi}_t - \alpha \gamma \boldsymbol{\phi}_{t+1} (\boldsymbol{\phi}_t^\top \mathbf{u}_t) \\ \mathbf{u}_{t+1} &\leftarrow \mathbf{u}_t + \beta(\delta_t - \boldsymbol{\phi}_t^\top \mathbf{u}_t) \boldsymbol{\phi}_t\end{aligned}$$

where α and β are learning rates which can also be set per time-step. This algorithm can also be derived when the value function is non-linear (Maei, Szepesvári, Bhatnagar, Silver, et al. 2009). See Chapter 5 for a non-linear derivation with added constraints.

2.3 Control in Reinforcement Learning

The control problem is the process of learning a policy which the agent can use to decide actions. There are many possible approaches for control in reinforcement learning, from value-based control to direct policy optimization through policy gradient and actor critic methods. All the control experiments in this thesis use value-based control as a means to study the perception of reinforcement learning agents (see 2.4 for more details).

As defined above, a state-action value function

$$Q^*(\psi, a) = \mathbb{E}_{\pi^*}[G_t | \Psi_t = \psi, A_t = a].$$

where π^* is the optimal policy is the main object for value based control. The goal of the agent is to search through the space of policies to maximize the total return the agent will receive from any state, or in other words to find the optimal policy π^*. In this thesis, our control experiments are restricted to Q-learning (Watkins and Dayan 1992; Mnih et al. 2015), an off-policy technique which learns the optimal policy. Q-learning, in its simplest form, is defined by the following set of updates

$$\begin{aligned} \delta_{t+1} &= R_{t+1} + \gamma \max_a (Q(\Psi_{t+1}, a)) - Q(\Psi_t, A_t) \\ Q(\Psi_t, A_t) &\leftarrow Q(\Psi_t, A_t) + \alpha \delta_{t+1} \end{aligned}$$

Similarly to the TD learning rule, semi-gradient updates can be derived from the tabular rule by minimizing an objective

$$\begin{aligned} U_t &= R_{t+1} + \gamma \max_a (Q(\Psi_{t+1}, a)) \\ \mathcal{L}(\Psi, A, \Psi', R_t) &= (\mathcal{Q}(\Psi, A; \mathbf{W}) - U_t)^2. \end{aligned}$$

So far, the above update rule for q-learning is defined for the tabular setting. In section 2.5, this algorithm is extended to the function approximation setting.

2.4 Perception and Partial Observability in Reinforcement Learning

In a setting which is partially observable the observations are a function of an unknown, unobserved underlying state. The dynamics, like in the fully observable case, are specified by transition probabilities $\mathrm{P} = \boldsymbol{\Psi} \times \mathcal{A} \times \boldsymbol{\Psi} \rightarrow [0, \infty)$ with state-space $\boldsymbol{\Psi}$ and action-space $\mathcal{A}$. On each time step the agent receives an observation vector $\mathbf{o}_t \in \mathcal{O} \subset \mathbb{R}^m$, as a function $\mathbf{o}_t = f_{\mathbf{o}}(\psi_t)$ of the underlying state $\psi_t \in \boldsymbol{\Psi}$. The agent only observes $\mathbf{o}_t \neq \psi_t$, and then takes an action a_t, producing a sequence of observations and actions: $\mathbf{o}_0, a_0, \mathbf{o}_1, a_1, \ldots$.

Observations do not have the Markov property to enable the use of the above reinforcement learning algorithms. Instead, the agent will define its functions through histories of observation action sequences. A history $h_k \overset{\text{def}}{=} (\mathbf{o}_0, a_0, \ldots, \mathbf{o}_{k-1}, a_{k-1}, \mathbf{o}_k)$ is said to have the Markov property if $p(\mathbf{o}_{k+1}|\mathbf{h}_k, a_k) = p(\mathbf{o}_{k+1}|\mathbf{o}_{-1}, a_{-1}, \mathbf{h}_k, a_k)$, where -1 indexing is meant to mean an observation that comes before the history. A history can be said to be minimal if it is the minimal history needed to ensure the Markov property $p(\mathbf{o}_{k+1}|\mathbf{h}_k, a_k) \neq p(\mathbf{o}_{k+1}|\mathbf{o}_1, a_1, \ldots, \mathbf{o}_{k-1}, a_{k-1}, \mathbf{o}_k, a_k)$. The minimal set of histories $\mathcal{H}$ enables the Markov property for the distribution over next observation

$$\mathcal{H} = \{\mathbf{h}_k \mid \text{ where } h_k \text{ has both the Markov and minimal properties. } \}. \quad (2.2)$$

The goal for the agent under partial observability is to identify a state representation $\mathbf{s}_t \in \mathbb{R}^N$ which is a sufficient statistic (summary) of history $\mathcal{H}$, for targets y_t. More precisely, such a *sufficient state* ensures that y_t given this state is independent of history $\mathbf{h}_t = \mathbf{o}_0, a_0, \mathbf{o}_1, a_1, \ldots, \mathbf{o}_{t-1}, a_{t-1}, \mathbf{o}_t$,

$$p(y_t|S_t) = p(y_t|S_t, \mathbf{h}_t) \quad (2.3)$$

or so that statistics about the target are independent of history, such as $\mathbb{E}[Y_t|\mathbf{s}_t] = \mathbb{E}[Y_t|\mathbf{s}_t, \mathbf{h}_t]$. Such a state summarizes the history, removing the need to store the entire (potentially infinite) history.

In the next two sections, I detail recurrent neural networks (RNNs) and the algorithms used to train RNNs in SL and RL.

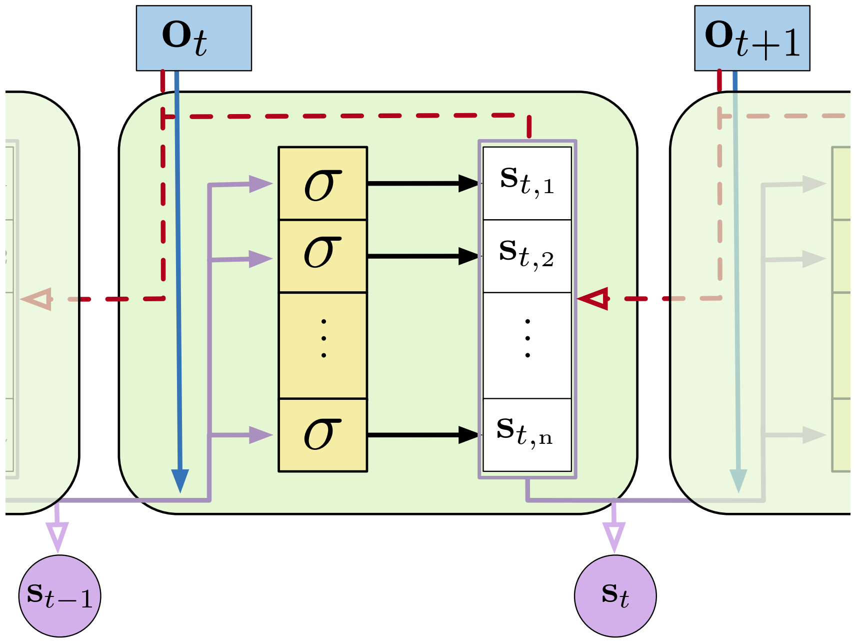


Figure 2.2: A visual representation of a simple recurrent network. The dotted lines denote gradients while the solid lines denote the forward operations both going through time.

2.4.1 Recurrent Neural Networks

Recurrent neural networks (RNNs) have been established as an important tool for learning predictions of data with temporal dependencies. They have been primarily used in language and video prediction (Mikolov et al. 2010; Oh et al. 2015; T. Wang and Cho 2016; Saon et al. 2017; Y. Wang et al. 2018), but have also been used in traditional time-series forecasting (Bianchi et al. 2017) and RL (Onat et al. 1998; Bakker 2002; Wierstra et al. 2007; Hausknecht and Stone 2015; Heess et al. 2015; Igl et al. 2018; Zhu et al. 2018). In this section, I will outline the three major architectures applied in this thesis. In the next section I will detail the algorithms deployed to train these architectures in SL and RL.

An RNN provides one such solution to learning $\mathbf{s}_t$ and the associated state update function. The simplest RNN is one which calculates state $\mathbf{s}_t \in \mathbb{R}^n$ recursively,

$$\mathbf{s}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{b})$$

where $\mathbf{x}_t = [\mathbf{o}_t, \mathbf{s}_{t-1}]$ and σ is any non-linear transfer function (typically tanh). While concatenating information (or doing additive operations) has become standard in RNNs, another idea explored

$$\mathbf{s}_{t,i} = \sigma\left(\sum_{j=1}^{M}\sum_{k=1}^{N}\mathbf{W}_{ijk}\mathbf{o}_{t,j}\mathbf{s}_{t-1,k} + \mathbf{b}_i\right) \qquad \triangleright \text{ where } \mathbf{W} \in \mathbb{R}^{|\mathbf{s}|\times|\mathbf{o}|\times|\mathbf{s}|}$$

where indexing into a vector on time-step t is indicated through a double subscript as $\mathbf{s}_{t,j}$ for the jth component of $\mathbf{s}_t$. Using this type of operation was initially called second-order RNNs (Goudreau et al. 1994), and was also explored in one of the first landmark successes of RNNs (Sutskever et al. 2011) in a character-level language modeling task.

There are several known problems with simple recurrent units (or simple recurrent cells). The first is known as the vanishing and exploding gradient problem (Pascanu et al. 2013). In this, as gradients are multiplied together (via the chain rule in back-propagation through time) the gradient can either become very large or vanish into nothing. In either case, the learned networks often cannot perform well and a number of practical tricks are applied to stabilize learning (Bengio et al. 2013). The second problem is called saturation. This occurs when the weights $\mathbf{W}$ become large and the activations of the hidden units are at the extremes of the transfer function. While not problematic for learning stability, this can limit the capacity of the network and make tracking changes in the environment dynamics more difficult (Chandar et al. 2019).

Many specialized architectures have been developed to improve learning with recurrence. These architectures are designed to better learn long-temporal dependences and avoid saturation (Hochreiter and Schmidhuber 1997; Cho et al. 2014; J. Chung et al. 2014; Greff et al. 2017; Chandar et al. 2019). The experiments presented in this work use three cell types. The first was the simple RNN introduced earlier in this section. The other cells used are Long short-term memory cells (LSTM) (Hochreiter and Schmidhuber 1997), and gated-recurrent units (GRU) (J. Chung et al. 2014) which are standard cells used throughout sequence prediction in supervised learning. Long short-term memory cells (LSTM) were developed to address the issues with modeling long-temporal dependencies and the vanishing gradients problem observed in simple RNN cells. Gated-recurrent units (GRU) are a modification from the

LSTM cell which maintains performance in many settings, improves ease of use, and improves computational footprint (Greff et al. 2017).

2.4.2 Back-Propagation Through Time

In supervised learning, back-propagation through time (BPTT) (Mozer 1995) is a commonly used algorithm for estimating the gradients of recurrent networks. In this section, BPTT is briefly introduced alongside some alternatives. This algorithm effectively unrolls the network through the sequence and calculates the gradient as if it was one large network with shared weights. When calculating the gradients through time for a specific sample using BPTT, the loss can be defined [3] as

$$\mathcal{L}_t(\mathbf{o}_1, \dots, \mathbf{o}_t, y_t, \mathbf{W}_t) = \sum_i^N (V_i(\mathbf{s}(\mathbf{o}_1, \dots, \mathbf{o}_t; \mathbf{W}_t)) - y_{t,i})^2$$

where N is the size of the batch, and y is the target defined by the specific algorithm, and t is the current time. This will calculate the loss for a single step at the end of the sequence rolling back through the entire sequence to the beginning.

One might notice the above loss function requires growing computational and memory requirements as the agent interacts with the environment. To limit the computational and memory concerns, the current standard in training recurrent architectures in RL is truncated back-propagation through time. p-BPTT truncates the unrolling of the network to some number of steps p. While this alleviates computational-cost concerns, the learning performance can be sensitive to the truncation parameter (Pascanu et al. 2013), particularly if the dependencies back-in-time are longer than the chosen p—as reaffirmed by the results in this thesis. The loss is slightly modified from above as

$$\mathcal{L}_t(\mathbf{o}_{t-p}, \dots, \mathbf{o}_t, y_t, \mathbf{W}_t) = \sum_i^N (V_i(\mathbf{s}(\mathbf{o}_{t-p}, \dots, \mathbf{o}_t; \mathbf{W}_t)) - y_{t,i})^2.$$

An alternative to p-BPTT is real time recurrent learning (RTRL) (R. J. Williams and Zipser 1989). Unfortunately RTRL is known to suffer high

[3]You can also sum over the temporal dimension, passing in a sequence of labels to the loss. In this work, the loss is calculated with respect to the most recent time-step (or the end of sampled sequence) and calculate gradients back from this.

computational costs for large networks. Several approximations have been developed to alleviate these costs (Mujika et al. 2018; Tallec and Ollivier 2018), but these algorithms often struggle from high variance updates making learning slow. The approximation to the RTRL influence matrix proposed by Menick et al. 2020 shows significant promise in sparse recurrent networks, even outperforming BPTT when trained fully online. Ke et al. (2018) propose a sparse attentive backtracking credit assignment algorithm inspired by hippocampal replay, showing evidence the algorithm has beneficial properties of both BPTT and truncated BPTT. The focused architecture was often able to compete with the fully connected architecture on length of learned temporal sequence and prediction error on several benchmark tasks. Another line of search/credit assignment algorithms is generate and test (Kudenko and Hirsh 1998; Mahmood and Sutton 2013; Samani and Sutton 2021; Dohare et al. 2022). These search algorithms aren't as tied to their initialization as other systems as they intermittently inject randomness into their search to jump out of local minima. Many of these approaches combine both gradient descent and generate and test to gain the benefits of both. While a full generate and test solution is possible, finding the right heuristics to generate useful state objects quickly could be problem dependent.

2.5 Prediction and Control in Deep Recurrent Reinforcement Learning

When applying deep recurrent neural networks to the RL setting, there are several components which have been shown improve learning (Mnih et al. 2015). In this section, these components are introduced in a piecemeal way. In the empirical results of this thesis, all details are provided closer to their respective results.

Loss Functions: In all the following results, semi-gradient learning updates are used unless otherwise specified. Given a loss function of the form

$$\mathcal{L}(S_t, A_t, S_{t+1}, R_t) = f(U_t, \mathcal{Q}(S_t, A_t; \mathbf{W}))$$

where the bootstrapped target is $U_t = R_t + \gamma \max_a \mathcal{Q}(S_{t+1}, a; \mathbf{W}_t)$ for Watkins q-learning (Watkins and Dayan 1992). The semi-gradient learning update only considers the gradient with respect to $\mathcal{Q}(S, A; \mathbf{W})$ where the gradient of the target $\nabla_{\mathbf{W}} U_t = 0$. The function f is the mean squared error $f(\hat{y}, y) = (\hat{y} - y)^2$, unless otherwise specified.

Experience Replay Buffer: The experience replay buffer is mechanism for re-using data (Lin 1992; Lin 1993) and for inducing an almost independent, identically distributed set of examples with which to train the network (Mnih et al. 2015; Schaul, Quan, et al. 2015). The replay buffer simply is a buffer of stored transitions $(S, A, S\prime, R)$ which is sampled according to some distribution (typically uniform).

Target Networks: Target networks are a slow moving copy of the network representing the q-function. It is updated according to pre-determined frequency of agent steps. The target-network is used to calculate the bootstrapped target (U_t above). This is said to provide a more stable target for the network to approximate (Mnih et al. 2015).

Auxiliary Tasks: Auxiliary tasks are a set of learning objectives unrelated to the underlying control problem used to supplement an often sparse reward structure. These tasks are often prediction tasks of the observations on a separate head of the network (Jaderberg et al. 2017). They have also been defined as a set of general value functions (see section 2.6) (Sutton, Modayil, et al. 2011; Jaderberg et al. 2017) and discovered through a meta-learning process (Veeriah et al. 2019) or generate-and-test (Rafiee, Ghiassian, et al. 2022).

2.5.1 Architectural Choices for Recurrent Networks in RL

Previously, the components of a deep feed-forward reinforcement learning agent were introduced. Many of these components are the same when the architecture is recurrent, but there are minor differences and challenges which are described below.

The woes of the experience replay buffer: Current deep learning, including recurrent architectures, in reinforcement learning include the need for an experience replay buffer. While a learning algorithm which overcomes this limitation would likely be preferable, in the short term cohesive strategies for combining an experience replay with recurrent architectures should be empirically explored. There are two major approaches currently: 1) using the stale traces, or 2) warming up the agent from the beginning (or some number of time-steps prior) of an episode (Hausknecht and Stone 2015). Instead, a third strategy is used in this thesis, we use gradient information to refresh the hidden state to minimize the objective. We found little difference between this and the stale approach. For much more insight and discussion on this choice (see Kapturowski et al. 2019).

Target networks and state: Using a recurrent target network introduces a new challenge. Specifically, in the approach chosen to initialize the hidden state of the target network. Several choices could be made such as rolling forward from the start of an episode, or using the state stored in the replay buffer generated when the example was originally seen. Another possible approach is to use the state in the buffer to regularize the learning of the network (Nath et al. 2020). Unless otherwise specified, the following results simply use the state stored in the buffer to initialize the recurrent network at the beginning of a sequence.

Objectives matter even more: It is known that some objective functions are more learnable in both the fully and partially observable settings (Mozer 1991; van Hasselt and Sutton 2015). In this thesis, a specific loss structure is adopted to make comparisons between several styles of architectures (see section 2.4.2 for details), but others could have been used. Another addition are auxiliary tasks which augment the objective function (Jaderberg et al. 2017), and has been argued to improve learning state representations.

2.6 Temporal Abstractions in Reinforcement Learning

Reinforcement learning is built on predicting the effect of behavior on future observations and rewards. Many of our algorithms learn predictions of a cumulative sum of (discounted) future rewards, which is used as a bedrock for learning desirable policies. While reward has been the primary predictive target of focus, TD models (Sutton 1995) lay out the use of temporal-difference learning to learn a world model through value function predictions. Temporal-difference networks (Sutton and Tanner 2005; Tanner 2005) take advantage of this abstraction and build state and representations through predictions. Sutton, Modayil, et al. (2011) and A. White (2015) further the predictive perspective by developing a predictive approach to building world knowledge through general value functions (GVFs).

Two objects in RL which enable agents to reason beyond the moment-to-moment stream of experience are known as **options** (Precup, Sutton, and Singh 1998) and **general value functions** (GVFs) (Sutton, Modayil, et al. 2011). In this thesis, I focus on applying GVFs to learning a representation of history, leaving the incorporation of options for future work. GVFs have been pursued broadly in reinforcement learning: Günther et al. (2016) used GVFs to build an open loop laser welder controller, Linke et al. (2020) and McLeod et al. (2021) used predictions and their learning progress to develop an intrinsic reward, Edwards et al. (2016) used GVFs to build controllers for myoelectric prosthetics, using GVFs for auxiliary training tasks to improve representation learning (Jaderberg et al. 2017; Veeriah et al. 2019), to extend a value function's approximation to generalize over goals as well as states (Schaul, Horgan, et al. 2015), and to create a scheduled controller from a set of sub-tasks for sparse reward problems (Riedmiller et al. 2018). Successor representations and features are predictions of the state, learned or given, which have been shown to improve learning performance (Dayan 1993; Russek et al. 2017; Barreto et al. 2018; Sherstan et al. 2018).

General value functions are a generalization of value functions enabling agents to learn value function predictions of their sensorimotor stream beyond a reward signal. A GVF question is a tuple (π, C, γ) composed of a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty)$, cumulant $C : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ and a continuation (or discount) function[4] $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ (M. White 2017). The answer to

[4]The original GVF definition assumed the continuation was only a function of S_{t+1}. This

a GVF question is a mapping $V : \mathcal{S} \to \mathbb{R}$ or $\mathcal{Q} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ to the expected discounted return of a cumulant function [5]:

$$V_{c,\gamma,\pi}(s) = \mathbb{E}\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k}\gamma(S_{i-1}, A_{i-1}, S_i)\right)c(S_{i-1}, A_{i-1}, S_i)\middle| S_t = s, A_{t:\infty} \sim \pi\right].$$

General value functions were introduced in the context of prediction (i.e. as a prediction demon). This object can be used in both prediction (as described above) and for control (similarly to options). A control demon is encoded into a state-action value function and learned through Q-learning (Watkins and Dayan 1992) or Sarsa (Rummery and Niranjan 1994) to maximize the return. This is similar to the control problem (Section 2.3)—where the objective is to use value iteration to learn a policy—but often the behavior policy is arbitrarily different from the current policy of the demon. While these objects present unique sets of predictive information, our focus in the thesis will be on prediction demons throughout the thesis and consider the incorporation of control demons in future work. GVF questions, or the definitions used here, can be used as a unifying specification for reinforcement learning tasks (i.e. for options, predictions, control, etc. . .) (M. White 2017).

was later extended to transition-based continuation (M. White 2017), to better encompass episodic problems. Namely, it allows for different continuations based on the transition, such as if there is a sudden change from S_t to S_{t+1}. I use this more general definition for this reason, and because the cumulant itself is already defined on the three tuple (S_t, a_t, S_{t+1}).

[5] Note how S is used instead of Ψ in these definitions. While the definition can be for either the agent state or the environment state, this dissertation focuses on the case when it is a function of agent state or history. See Chapter 4 for more details on GVFs defined on history.

Chapter 3

Incorporating Action into a Recurrent Network

Before we address the research question posed in this thesis, first we take a deeper look at how recurrent architectures are used in reinforcement learning and how we can improve them. One important design decision is the strategy used to incorporate action in the state update function which can have a large impact on the agent's ability to predict and control (see Figure 3.1). This has been noted before, Zhu et al. 2018 provides a discussion on the importance of these choices developing an architecture which encodes the action through several layers before concatenating with the observation encoding. Other types of action encodings have been used for the state update in RNNs for RL (Schaefer et al. 2007; Zhu et al. 2018; Schlegel, Jacobsen, et al. 2021), but without an in-depth discussion or focus on the ramifications of the particular choice of architecture. In other cases, action has seemingly been omitted (Hausknecht and Stone 2015; Oh et al. 2015; Espeholt et al. 2018). Other state construction approaches also see action as a primary component, predictive representations of state encode predictions as the likelihood of seeing action-observation pairs given a history (Littman and Sutton 2002).

The major contribution of this chapter is the comparison and analysis of several architectures for incorporating action into the state-update function of an RNN in partially observable RL settings. Many of these architectures have been proposed previously for recurrent architectures (i.e. Zhu et al. 2018; Schlegel, Jacobsen, et al. 2021), and others are either related to or obvious

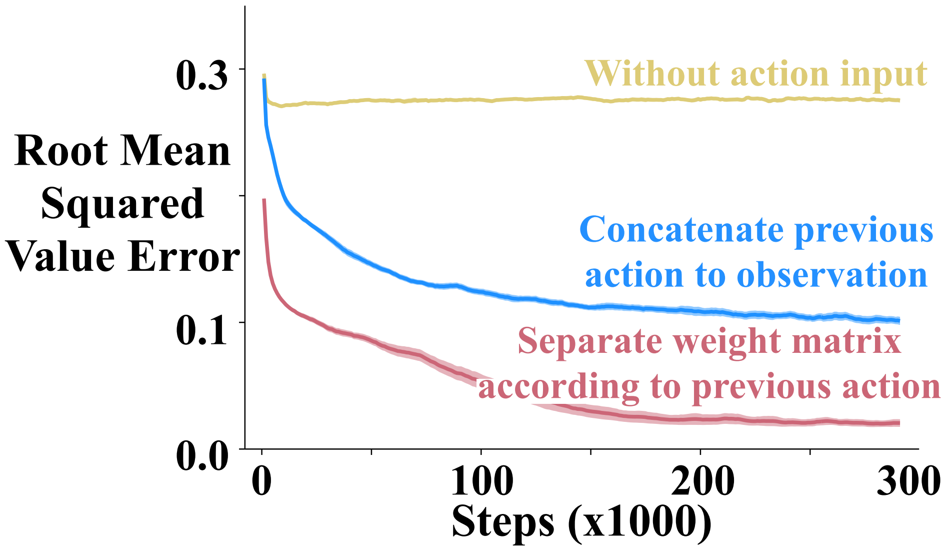


Figure 3.1: Learning Curves for various RNN cells in Ring World using experience replay and three strategies to incorporate action into an RNN. The agent learns 20 GVF predictions for 300k steps. The results reported is the root mean squared value error averaged over 50 runs with 95% confidence intervals with window averaging over 1000 steps. See Section 3.5 for full details.

extensions of those architectures. The results include an in-depth empirical evaluation on several illustrative domains, and outline the relationship between the domain and architectures using the deep recurrent q-network (DRQN) framework (Hausknecht and Stone 2015).

3.1 Constructing State with Recurrent Networks

For convenience, this section reiterates some of the details used to learn a state-update function using recurrent neural networks. Much of the content is the same as found in Section 2.4, with details specific to this contribution. Specifically, the details needed for incorporating RNNs into the deep Q-network framework as originally discussed by Hausknecht and Stone (2015).

For effective prediction and control, the agent requires a state representation $\mathbf{s}_t \in \mathbb{R}^n$ that is a sufficient statistic of the past: $\mathbb{E}\left[G_t^c|\mathbf{s}_t\right] = \mathbb{E}\left[G_t^c|\mathbf{s}_t, \mathbf{h}_t\right]$. When the agent learns such a state, it can build policies and value functions without the need to store any history. For example, for prediction, it can learn $V(\mathbf{s}_t) \approx \mathbb{E}\left[G_t^c|\mathbf{s}_t\right]$.

An RNN provides one such solution to learning $\mathbf{s}_t$ and associated state update function. The simplest RNN is one which learns the parameters $\mathbf{W} \in \mathbb{R}^d$ recursively

$$\mathbf{s}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{b})$$

where $\mathbf{x}_t = [\mathbf{o}_t, \mathbf{s}_{t-1}]$ and σ is any non-linear transfer function (typically *tanh*). In this chapter we use truncated back-propagation through time (p-BPTT) with details provided in section 2.4.

To improve sample efficiency experience replay (ER) is incorporated. ER is a critical part of a deep (recurrent) system in RL (Hausknecht and Stone 2015; Mnih et al. 2015). There are two key choices here: how states are stored and updated in the buffer and how sequences are sampled (Kapturowski et al. 2019). In the following sections, the hidden state of the cell is stored in the experience replay buffer as apart of the experience tuple. This is then used to initialize the state when sampled from the buffer for both the target and non-target networks. Gradients are passed back to the stored state to update them along with our model parameters, see a full discussion in Section 10.2.1. A separate initial state is also stored for the beginning of episodes, which is updated with gradients. This is slightly differ from the approach taken by Kapturowski et al. 2019, but this architectural choice should have little impact on the results presented here. If the beginning of an episode is sampled from the replay, the most up to date version of this vector was used to initialize the hidden state. The agent samples states across the episode. For samples at the end of the episode, a shorter sequence length τ is used.

3.2 Tensors and Low-Rank Decompositions

Before getting to the details of how to encode actions in the state-update function, I will provide the required background on Tensors. The simplest, albeit slightly inaccurate, way to describe and use a tensor is as a multi-dimensional array of numbers (either real or complex) which transform under coordinate changes in predictable ways. In this chapter, tensors are multi-dimensional arrays using Einstein summation notation. The ith, jth, kth component of an order-3 tensor will be denoted with lower indices $\mathbf{W}_{ijk} \in \mathbb{R}$ with associated dimension size denoted with corresponding uppercase letters as $\mathbf{W} \in \mathbb{R}^{I \times J \times K}$.

Like matrices, tensors have a number of decompositions which can prove useful. For example, every tensor can be factorized using canonical polyadic decomposition (CP decomposition), which decomposes an order-N tensor $\mathbf{W} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ into N matrices as follows

$$\begin{aligned}\mathbf{W}_{i_1,i_2,\ldots} &= \sum_{r=1}^{R} \lambda_r \mathbf{W}^{(1)}_{i_1,r} \mathbf{W}^{(2)}_{i_2,r} \ldots \mathbf{W}^{(N)}_{i_N,r} \\ &= \lambda_r \mathbf{W}^{(1)}_{i_1,r} \mathbf{W}^{(2)}_{i_2,r} \ldots \mathbf{W}^{(N)}_{i_N,r} \quad \triangleright \text{Explicit summation over } r \in \{1, \ldots, R\}.\end{aligned}$$

where $\mathbf{W}^{(j)} \in \mathbb{R}^{I_j \times R}$, and R is the rank of the tensor. This is a generalization of matrix rank decomposition, and exists for all tensors with finite dimensions.

Working with tensors takes a bit more care in deciding which fibers (generalization of row and column) the product should be over. One type of product is known as the n-mode product which is defined as follows

$$(\mathbf{W} \times_n \mathbf{v})_{i_1,i_2,\ldots,i_{n-1},j,i_{n+1},\ldots i_N} = \mathbf{W}_{i_1,i_2,\ldots,i_{n-1},i_n,i_{n+1},\ldots i_N} \mathbf{v}_{j,i_n}$$

where $\mathbf{v} \in \mathbb{R}^{J,I_n}$.

An important property, which will be used later in this chapter, are the simplifications when using n-mode products with a tensor's rank decomposition. For example, order 3 tensors $\mathbf{W} \in \mathbb{R}^{IJK}$, with CP-decomposition $\mathbf{W}_{ijk} = \lambda_r a_{ir} b_{jr} c_{kr}$ and vector over a strand $\mathbf{v}^M = \mathbf{v}^{(1,M)} \in \mathbb{R}^{1 \times M}$).

$$\begin{aligned}(\mathbf{W} \times_2 \mathbf{v}^J \times_3 \mathbf{v}^K)_{i,1,1} &= \sum_{k=1}^{K} \left(\sum_{j=1}^{J} \mathbf{W}_{ijk} \mathbf{v}^J_{1j} \right) \mathbf{v}^K_{1k} \\ &= \sum_{k=1}^{K} \sum_{j=1}^{J} \left(\sum_{r=1}^{R} \lambda_r a_{ir} b_{jr} c_{kr} \right) \mathbf{v}^J_{1j} \mathbf{v}^K_{1k} \\ &= \sum_{r=1}^{R} \lambda_r a_{ir} \left(\sum_{j=1}^{J} b_{jr} \mathbf{v}^J_{1j} \right) \left(\sum_{k=1}^{K} c_{kr} \mathbf{v}^K_{1k} \right) \\ &= \sum_{r=1}^{R} \lambda_r a_{ir} \left(\mathbf{v}^J \mathbf{B} \odot \mathbf{v}^K \mathbf{C} \right)_{1r} \\ \mathbf{W} \times_2 \mathbf{v}^J \times_3 \mathbf{v}^K &= \boldsymbol{\lambda} \mathbf{A} \left(\mathbf{v}^J \mathbf{B} \odot \mathbf{v}^K \mathbf{C} \right)^\top \quad \triangleright \boldsymbol{\lambda}_{i,i} = \lambda_i\end{aligned}$$

Similarly to CP decomposition, Tucker rank decomposition can be used to create a similar operation. Tucker rank decomposition decomposes an order-N tensor $\mathbf{W} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ into N matrices another order-N tensor $G \in \mathbb{R}^{R_1 \times R_2 \times \ldots \times R_N}$ as follows

$$\mathbf{W}_{i_1,i_2,\ldots i_N} = \sum_{r_1=1}^{R_1} \sum_{r_1=1}^{R_1} \cdots \sum_{r_1=1}^{R_1} g_{r_1 r_2 \ldots r_N} \mathbf{W}^{(1)}_{i_1,r_1} \mathbf{W}^{(2)}_{i_2,r_2} \ldots \mathbf{W}^{(N)}_{i_N,r_N}.$$

With similar simplifications to CP decomposition,

$$\begin{aligned}
(\mathbf{W} \times_2 \mathbf{v}^J \times_3 \mathbf{v}^K)_{i,1,1} &= \sum_{k=1}^{K} \left(\sum_{j=1}^{J} \mathbf{W}_{ijk} \mathbf{v}^J_{1j} \right) \mathbf{v}^K_{1k} \\
&= \sum_{k=1}^{K} \sum_{j=1}^{J} \left(\sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_{ip} b_{jq} c_{kr} \right) \mathbf{v}^J_{1j} \mathbf{v}^K_{1k} \\
&= \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_{ip} \left(\sum_{j=1}^{J} b_{jq} \mathbf{v}^J_{1j} \right) \left(\sum_{k=1}^{K} c_{kr} \mathbf{v}^K_{1k} \right) \\
&= \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_{ip} \left(\mathbf{v}^J \mathbf{B} \right)_{1q} \left(\mathbf{v}^K \mathbf{C} \right)_{1r} \\
\mathbf{W} \times_2 \mathbf{v}^J \times_3 \mathbf{v}^K &= G \times_1 \mathbf{A}^\top \times_2 \left(\mathbf{v}^J \mathbf{B} \right)^\top \times_3 \left(\mathbf{v}^K \mathbf{C} \right)^\top \\
&= \mathbf{A} \left[\left(G^\top \times_2 \left(\mathbf{v}^J \mathbf{B} \right)^\top \right) \left(\mathbf{v}^K \mathbf{C} \right)^\top \right].
\end{aligned}$$

One interesting property of this operation is now each of the dimensions can have a separately tuned rank, giving the system designer more discretion on where to focus representational resources.

A low-rank approximation of a multiplicative operation has been used several times before. A multiplicative update was used to make action-conditional video predictions in Atari (Oh et al. 2015). This operation also appears in a low-rank approximation defined by Predictive State RNN hidden state update (Downey et al. 2017), albeit the low-rank approximation never performed as well as the full rank version.

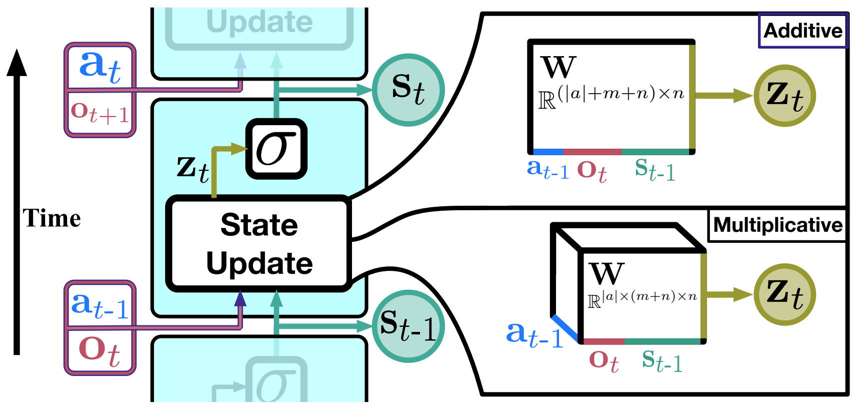

Update
$\mathbf{a}_t$
$\mathbf{o}_{t+1}$
$\mathbf{s}_t$
σ
$\mathbf{z}_t$
Time
State Update
$\mathbf{a}_{t\text{-}1}$
$\mathbf{o}_t$
$\mathbf{s}_{t\text{-}1}$
Additive
$\mathbf{W}$
$\mathbb{R}^{(|a|+m+n)\times n}$
$\mathbf{z}_t$
$\mathbf{a}_{t\text{-}1}$ $\mathbf{o}_t$ $\mathbf{s}_{t\text{-}1}$
Multiplicative
$\mathbf{W}$
$\mathbb{R}^{|a|\times(m+n)\times n}$
$\mathbf{z}_t$
$\mathbf{a}_{t\text{-}1}$
$\mathbf{o}_t$ $\mathbf{s}_{t\text{-}1}$

Figure 3.2: Visualizations of the multiplicative and additive RNNs. The dimensions of the weight matrices use the size of the RNN's state $|s_{t-1}| = n$ and the size of the observation $|o_t| = m$.

3.3 Architectural Designs for Incorporating Action

There are two broad categories for incorporating action into the state update function of an RNN. This section defines these categories and discusses various variations in these categories (see Figure 3.2 for a visualization of two main architectures).

3.3.1 Additive

The first category is to use an additive operation. The core concept of additive action recurrent networks is concatenating an action embedding as an input into the recurrent cell (Schaefer et al. 2007; Zhu et al. 2018). For example, the update becomes

$$\mathbf{s}_t = \sigma\left(\mathbf{W}^{\mathbf{x}}\mathbf{x}_t + \mathbf{W}^{\mathbf{a}}\mathbf{a}_{t-1} + \mathbf{b}\right) \qquad \textbf{(Additive)}$$

where $\mathbf{W}^{\mathbf{x}}$ and $\mathbf{W}^{\mathbf{a}}$ are appropriately sized weight matrices. This requires no changes to the recurrent cell if the action embedding $\mathbf{a}_{t-1} \in \mathbb{R}^b$ is concatenated to the observation vector. In the empirical experiments, the additive update cells use a hand-designed one-hot encoding function as all our domains have discrete actions.

A variant of the additive approach was explored in Zhu et al. 2018, where they modified the architecture slightly to learn a function of the action input

$\mathbf{a}_t = f_a(a_t)$. The label **Deep Additive** is used for this architecture, where the action encoding function f_a is a feed-forward neural network. As in their architecture, the action embedding is concatenated with the observation encodings right before the recurrent network. This focuses the empirical evaluation on the changes in the basic operation rather than enumerating all possible places the action can be concatenated before the recurrent operation.

3.3.2 Multiplicative

The second category is inspired by second-order RNNs (Goudreau et al. 1994) and first appeared as a part of a state update function in Rafols, Koop, et al. 2006, where the observation, hidden state, and action embedding are integrated using a multiplicative operation:

$$\mathbf{s}_t = \sigma\left(\mathbf{W} \times_2 \mathbf{x}_t \times_3 \mathbf{a}_{t-1}\right), \qquad \textbf{(Multiplicative)}$$

where $\mathbf{W} \in \mathbb{R}^{|\mathbf{s}_t| \times |\mathbf{x}_t| \times |\mathbf{a}_{t-1}|}$ and $\times_n$ is the n-mode product, which is detailed in Section 3.2. This type of operation is known to expand the types of functions learnable by a single layer RNN (Goudreau et al. 1994; Sutskever et al. 2011), and decreases the network's sensitivity to truncation (Schlegel, Jacobsen, et al. 2021).

While this type of update has very clear advantages, there is also a tradeoff in terms of number of parameters and potential re-learning depending on the granularity of the action representation. For example, in the Ring World experiment depicted in Figure 3.1 the RNN cell with additive used 285 parameters with hidden state size of 15. The multiplicative version would have used 510 parameters with the same hidden state size. While this doesn't seem like a lot, in a domain like Atari (with 18 actions, 1024 inputs, and $|s_t| = 1024$) the number of parameters would be 2 million vs 38 million respectively. As shown below in the empirical study, the size of the state can be significantly reduced when using a multiplicative update. In any case, it would be worthwhile to develop strategies to reduce the number of parameters, which is discussed next.

3.3.3 Reducing Parameters of the Multiplicative

The first way to reduce the number of parameters is by using a low-rank approximation of the tensor operations. Like matrices, tensors have a number of decompositions which can prove useful. For example, every tensor can be factorized using canonical polyadic decomposition, which decomposes an order-N tensor $\mathbf{W} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ into n matrices as follows

$$\mathbf{W}_{i_1,i_2,\ldots} = \sum_{r=1}^{M} \lambda_r \mathbf{W}^{(1)}_{i_1,r} \mathbf{W}^{(2)}_{i_2,r} \ldots \mathbf{W}^{(N)}_{i_N,r}$$

where $\mathbf{W}^{(j)} \in \mathbb{R}^{I_j \times M}$, $\lambda_r \in \mathbb{R}$ is the weighting for factor r, and M is the rank of the tensor. This is a generalization of matrix rank decomposition and exists for all tensors with finite dimensions, see Appendix 3.2 for more details. Several simplifications using the properties of n-mode products can be made. Using the definition of the multiplicative RNN update,

$$\mathbf{W} \times_2 \mathbf{x}_t \times_3 \mathbf{a}_{t-1} \approx \boldsymbol{\lambda} \mathbf{W}^{out} \left(\mathbf{x}_t \mathbf{W}^{in} \odot \mathbf{a}_{t-1} \mathbf{W}^{a} \right)^\top \quad \triangleright \boldsymbol{\lambda}_{i,i} = \lambda_i. \quad \textbf{(Factored)}$$

Previous work explored using a low-rank approximation of a multiplicative operation. A multiplicative update was used to make action-conditional video predictions in Atari (Oh et al. 2015). This operation also appears in a Predictive State RNN hidden state update (Downey et al. 2017), albeit it never performed as well as the full rank version. Our low rank approximation is also similar to the network used in Sutskever et al. 2011, where they mention optimization issues (which were overcome through the use of quasi-second order methods).

Another approach to reducing the number of parameters required—and to reduce redundant learning—by using an action embedding rather than a one-hot encoding. For example, in Pong it is known that only ~5 actions matter. By taking advantage of the structure of the action space further reductions to the number of parameters could be made. This architecture is explored briefly in Section A.3. While this is an important piece of the puzzle, no effort is afforded to learning good action embeddings in the following results and I leave it to future work.

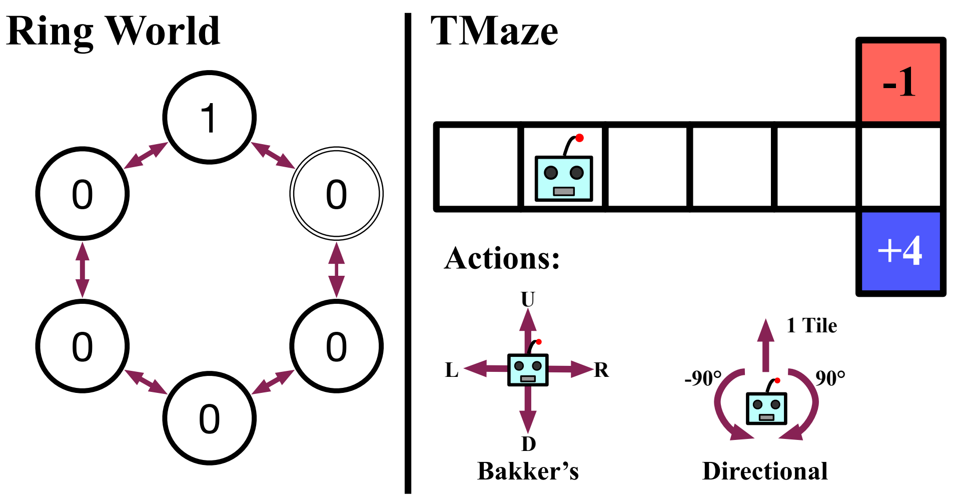


Figure 3.3: The illustrative environments used in Section 3.5 and Section 3.6 respectively. (**left**) The Ring World environment with 6 states is depicted, where the observation the agent receives is denoted in each of the circles, available actions denoted by the red arrows, and the agent's current location denoted by a double line. (**right**) The base TMaze environments are depicted with the available actions denoted below and labeled according to the Bakker's TMaze and Directional TMaze used in Section 3.6.

3.4 Empirical Questions

In the following sections, I set out to empirically evaluate the three operations for incorporating action into the state update function: **N**o **A**ction input ("**NA**"), **A**dditive **A**ction ("**AA**"), **M**ultiplicative **A**ction ("**MA**"), **Fac**tored ("**Fac**"), **D**eep **A**dditive **A**ction ("**DAA**"). I explore all the variants using both standard RNNs and a GRU cell. Our experiments are primarily driven by the main hypothesis that the multiplicative will strictly outperform the other variants. To explore this hypothesis I focus on two main empirical questions:

1. How do the different cells affect the properties of the learned value function and internal state of the agent?
2. Are there examples where the other variants outperform the multiplicative variant?

Question 1:

There are several properties I am interested in when analyzing the learning capabilities of the different architectures. First, and most obvious, is prediction error (calculated using root mean squared value error). While error is a reasonable method to compare different architectures, Kearney and Oxton (2019) argue only inspecting error can be misleading in the quality of the prediction. To account for this in our analysis I visually inspect the raw predictions as well to confirm they are reasonably modeling the target returns. With respect to the internal state, I am primarily interested in understanding if there are qualitative differences which lead to differences in prediction quality.

Question 2:

The second question is more straightforward than the first, and requires a complete empirical investigation of all the variants on a set of problems with a diverse set of underlying dynamics and characteristics. You can see this question as an extension of the hypothesis implied by Figure 3.1:

> The multiplicative update outperforms the other variants in the reinforcement learning setting for both control and prediction.

While the above hypothesis cannot be confirmed empirically, if question 2 is affirmed then the hypothesis is false. Counter examples for the hypothesis will also lead to more intuitive knowledge about when to apply one of the above variants.

3.4.1 Experimental Setup:

In all control experiments, an ϵ-greedy policy with $\epsilon = 0.1$ is used. All networks are initialized using a uniform Xavier strategy (Glorot and Bengio 2010), with the multiplicative operation independently normalizing across the action dimension (i.e. each matrix associated with an action in the tensor is independently sampled using the Xavier distribution). Unless otherwise stated, a hyperparameter search was performed for all models using a grid search over various parameters (listed appropriately in the Appendix A.4). The number of hyperparameter settings were fixed to be equivalent across all models, except the factored variants which use several combinations of hidden state size and number of factors. The best settings were selected and reported using independent runs with seeds different from those used in the hyperparameter

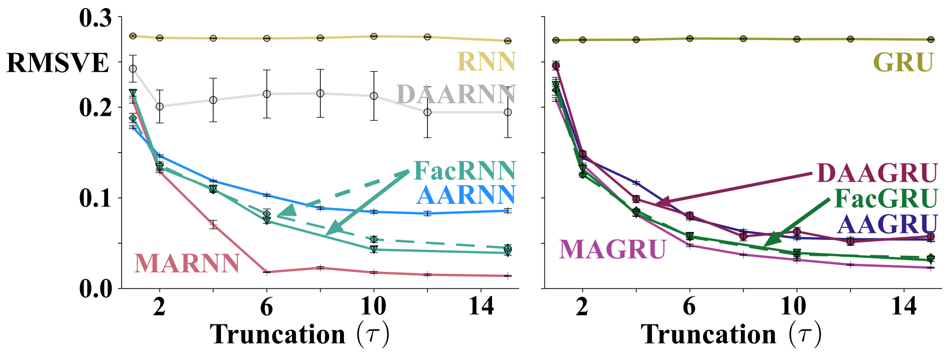


Figure 3.4: Ring World sensitivity curves of RMSVE over the final 50k steps for CELL (hidden size) **(left)** RNN (15), AARNN (15), MARNN (12), FacRNN (12 [solid] and 15 [dashed]), DARNN (12, $|\mathbf{a}| = 2$), and **(right)** GRU (12), AAGRU (12), MAGRU (9), FacGRU (9 [solid] and 12 [dashed]), DAGRU (9, $|\mathbf{a}| = 10$). Reported results are averaged over 50 runs with a 95% confidence interval. FacRNN used factors $M = \{12, 8\}$ respectively, and FacGRU used $M = \{14, 12\}$. All agents were trained over 300k steps.

search, unless otherwise specified. All the network sizes were controlled such that they had an approximately equal number of free parameters. All final network sizes can be found in Appendix A.4.

3.5 Investigating Properties of the Predictions and State

I explore the first empirical question by revisiting the Ring World environment, specifically to test model performance with various truncations, and to compare the architecture's learned state. The Ring World, depicted in Figure 3.3, consists of a cycle of states with a single state containing an active observation bit, and other states having an inactive observation bit. The agent can take actions moving either clockwise or counter clockwise in the cycle of states. The agent must keep track of how far it has moved from the active bit. For all experiments, Ring World had 10 underlying states.

The agent's objective is to learn a total of 20 GVFs with state-termination continuation functions of $\gamma \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. When the agent observes the active bit in Ring World (i.e. enters the first state) the predictions are terminated (i.e. $\gamma = 0.0$). The GVFs use the observed bit as a cumulant. Half of the GVFs follow a persistent policy of going clockwise and the other follow the opposite direction persistently. The agent follows an equiprobable random behavior policy. The agent updates its weights on every step following a off-policy semi-gradient TD update with a truncation values denoted. The agent was trained for 300000 steps and averaged over 50 independent runs. Root mean squared value error (RMSVE) is used as the main error metric. RMSVE can be calculated as

$$\text{RMSVE}_t = \frac{1}{|V(\mathbf{h}_t)|}||V(\mathbf{h}_t) - V_{\text{oracle}}(\psi_t)||_2,$$

where V_{oracle} is a known oracle for the true value function.

3.5.1 Results

Figure 3.4 contains a survey over truncation values for all the architectures. For both the RNN and GRU cells the MA variant performs the best, while the additive performs the worst of the cells which include action information. Interestingly, the factored variants for the GRU perform almost identically, while the FacRNN with a smaller hidden state perform marginally better. All factored variants straddled the performance of the additive and multiplicative updates. The DAAGRU performs similarly to the AAGRU, while the DAARNN fails to learn in this setting. Finally, the MARNN performs the best overall, only needing a truncation value of $\tau = 6$ to learn, which is shorter than the Ring World. With the same number of parameters, the operation used to update the state can have a significant effect on the required training sequence length τ and final performance.

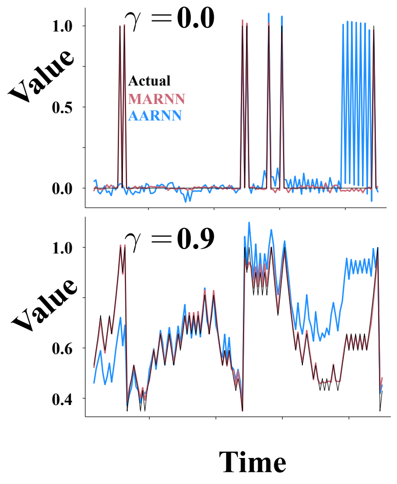


Figure 3.5: Ring World predictions of seed = 62 for the multiplicative and additive RNNs. Discounts listed with the target policy persistently going counter-clockwise.

To ground the prediction error reported, two representative examples of the learned predictions for the additive and multiplicative RNNs are reported in

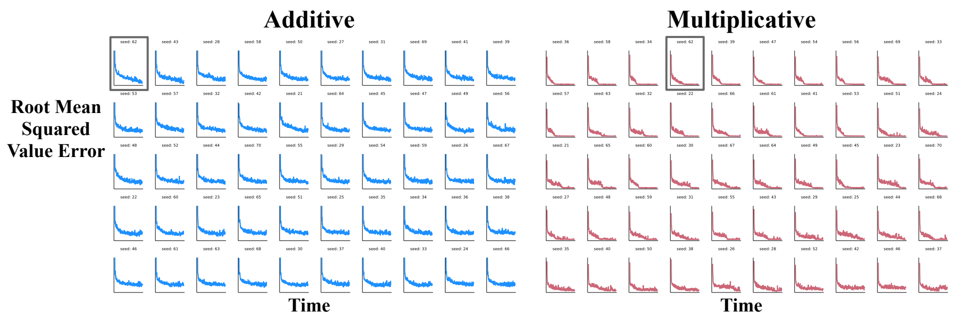


Figure 3.6: Individual learning curves for the additive (hidden size of 15) and multiplicative (hidden size 12) RNNs in Ring World with truncation $\tau = 6$. The plots are smoothed with a moving average with 1000 step window sizes. The gray box denotes the seed used in Figures 3.5 and 3.7. Overall, the multiplicative is quite resilient to initialization, but the distance from zero error in Figure 3.1 can be explained by a few bad initializations.

Figure 3.5. These plots show a single seed (selected as the best for the additive) over a small snippet of time, but are representative of our observations of the general performance for both cells. The multiplicative follows the actual prediction within a small delta being as close to zero error as can be expected, while the additive has many artifacts and other miss-predictions for both the myopic ($\gamma = 0.0$) and long-horizon ($\gamma = 0.9$) predictions. In Figure 3.6, all the individual learning curves were reported for the additive and multiplicative.

3.5.2 Looking Beyond Performance

A natural question is why might the multiplicative cell perform significantly better than the other cells in this simple setting? One hypothesis is that the multiplicative cell does a better job at separating the histories on action sequence as compared to the additive operation. While this question is difficult to test, one can peer into the learned state of each cell and see if there are qualitative features that appear to help explain the better performance. After learning (using the same parameters as in Figure 3.4) another 1000 steps of hidden states are collected. With these hidden states TSNE (Maaten and Hinton 2008) is applied to reduce the space of hidden states to two dimensions. The resulting scatter plots for the additive and multiplicative simple RNNs can be seen in Figures 3.7 and 3.8.

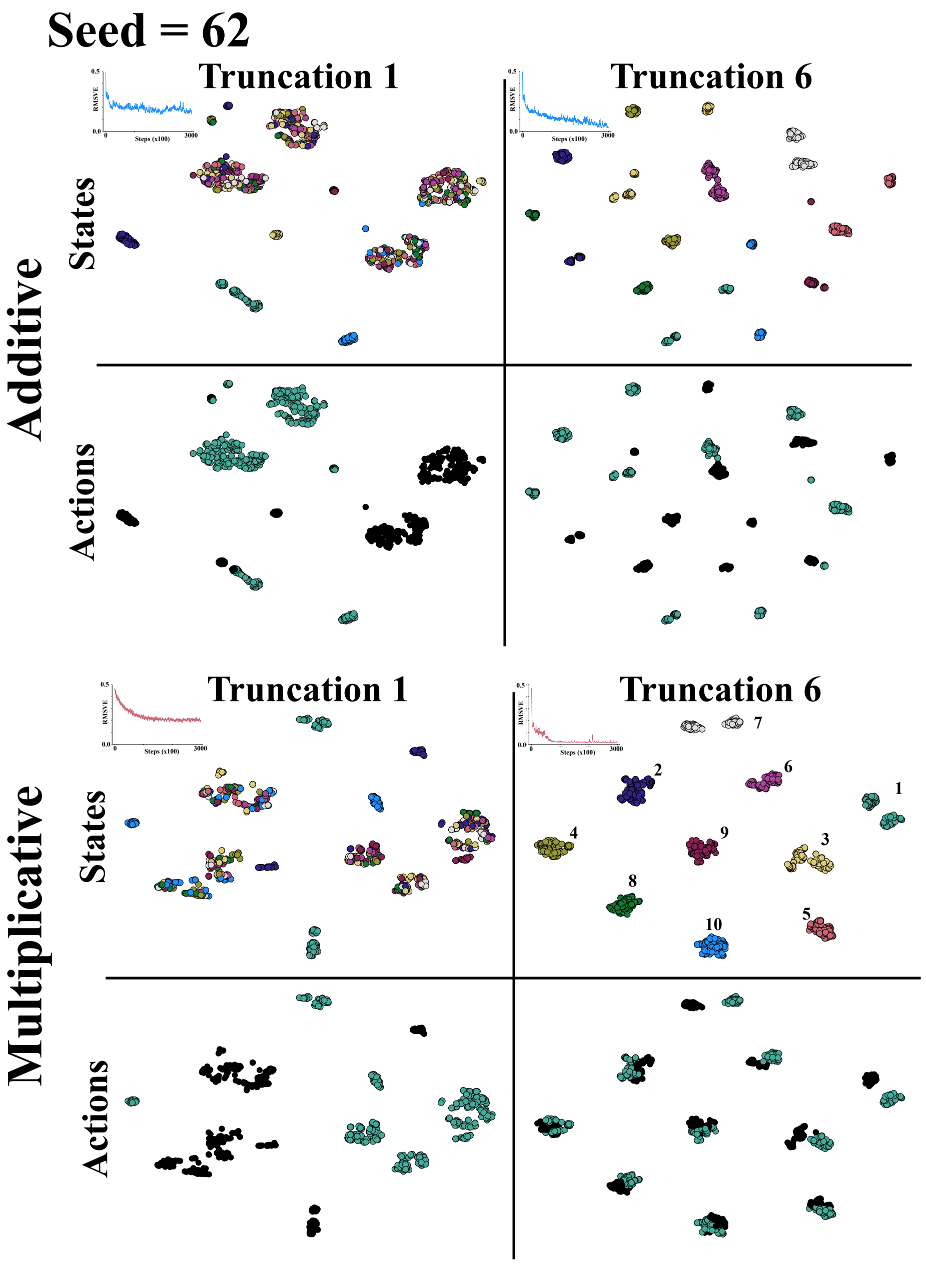


Figure 3.7: TSNE plots for the additive and multiplicative RNNs for truncation $\in \{1, 6\}$. Given the learning objective (described in Section 3.5), the state should have 10 distinct clusters for each state of the underlying environment. One would expect the truncation $\tau = 1$ to not be able to produce this kind of state for either cell variant. The learning curves correspond to a single seed (seed=62 which is best for the Additive update). The top scatter plots are colored on the underlying state the agent is currently in, the bottom scatter plots are colored based on the previous action the agent took. TSNE is initilized with the same random seed, with max iterations set to 1000, and perplexity set to 30. **(top)** additive and **(bottom)** multiplicative update functions.

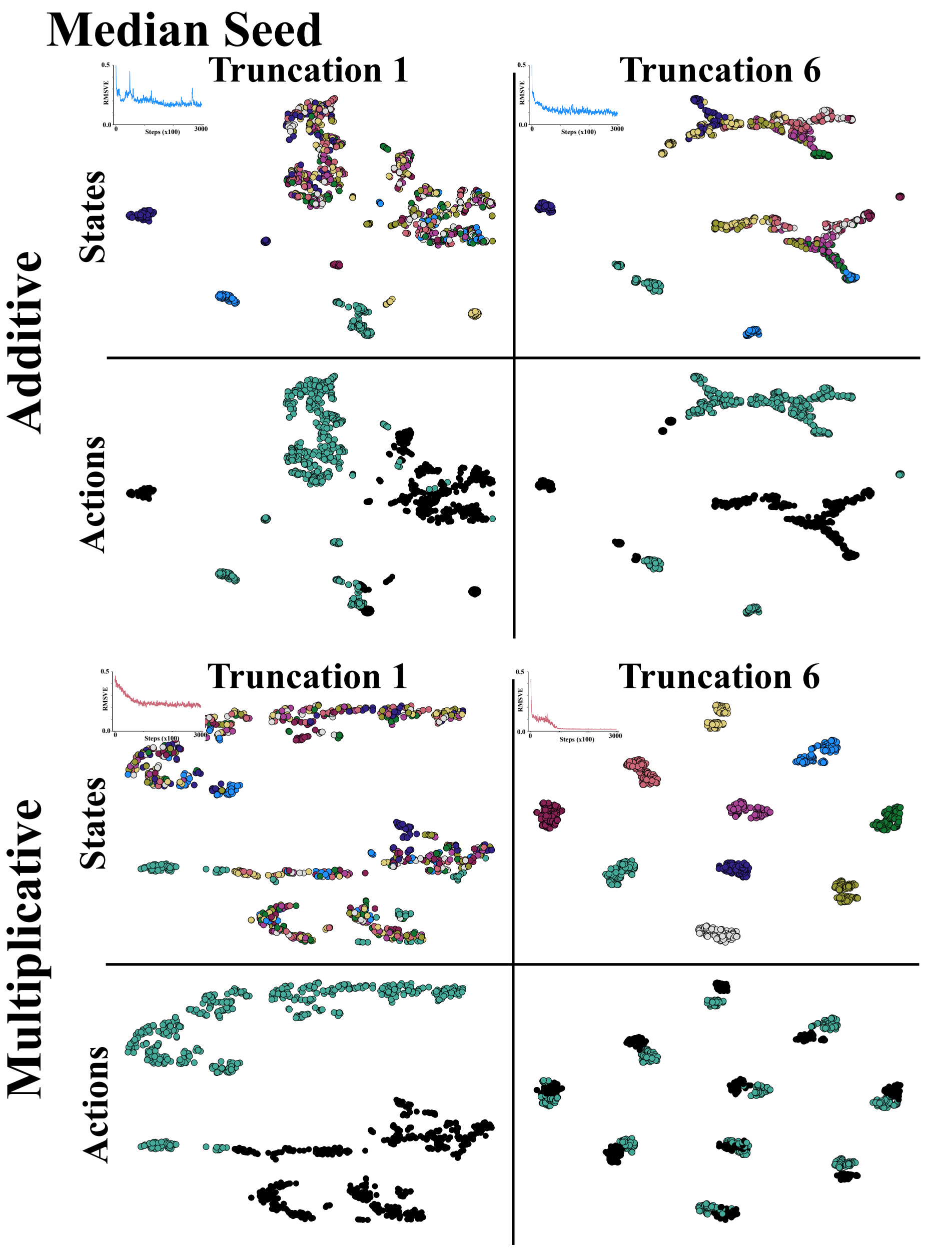


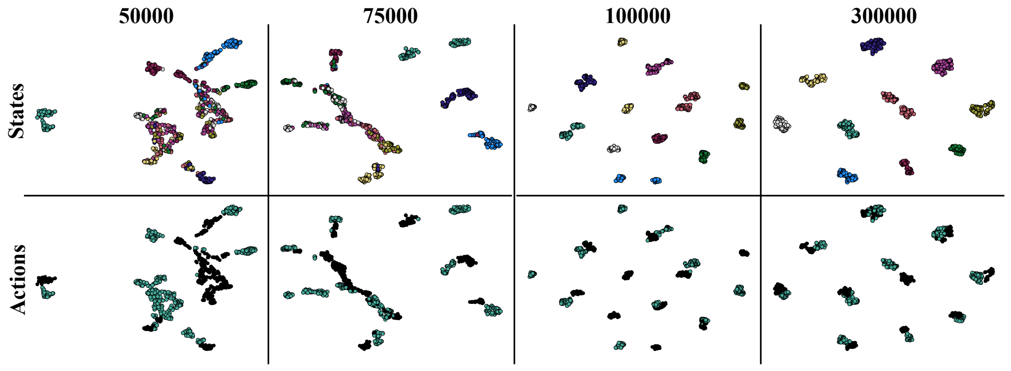
Figure 3.8: TSNE plots for the additive and multiplicative RNNs for truncation $\in \{1, 6\}$. Given the learning objective (described in Section 3.5), the state should to have 10 distinct clusters for each state of the underlying environment. One would expect the truncation $\tau = 1$ to not be able to produce this kind of state for either cell variant. The learning curves correspond to a single seed. The top scatter plots are colored on the underlying state the agent is currently in, the bottom scatter plots are colored based on the previous action the agent took. TSNE is initialized with the same random seed, with max iterations set to 1000, and perplexity set to 30. The results are presened for the median seeds of both cells **(top)** additive uses seed=55 and **(bottom)** multiplicative uses seed=67.

Overall, the additive and multiplicative's states can be separated on the previous action equally well, matching our initial hypothesis. While action is important, the additive seems to be hyper-focused on action even as the cell is able to partition on environment state. The multiplicative, on the other hand, is able to cluster the hidden states for various environment states together with only minor separation on action as seen in states 1 and 7. It is possible this is a natural part of th learning process for both the cells, but the multiplicative is able to cluster the states in less samples. Looking at the median performer (seed=55 and seed=67 for the additive and multiplicative respectively), the additive fails to separate on environment state, while the multiplicative looks similarly to the previous seed.

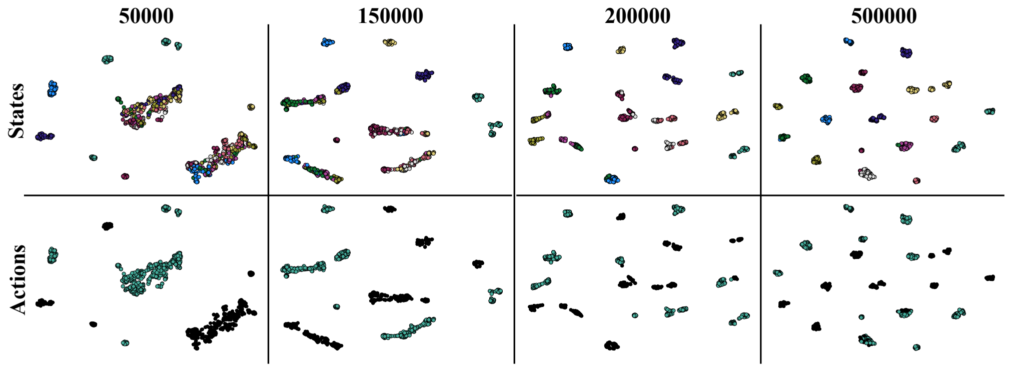
Above, I hypothesized the separation of action faced by the additive agent could have been an artifact of the learning dynamics. To test this hypothesis TSNEs were generated for several agents at different points in the training process. The results can be seen in Figure 3.9. For the multiplicative [50000, 75000, 100000, 300000] are reported. These temporal points represent the major learning milestones of the network. For the additive [50000, 150000, 200000, 500000] was chosen. These go beyond the original experiment's sample limits and shows the major milestones when the network separates the histories according to state. For 100000 steps of training for the multiplicative similar properties where the actions taken to get to specific states are quite separated. As the number of samples grow, to 300000, the states converging to be mostly clustered together regardless of the action taken. The additive version never sees the states converging, where even after 500000 time-steps the actions are still regarded highly by the network.

3.6 Understanding when Action Encoding Does and Does Not Matter

This section investigates learning behavior in two environments with slightly differing properties. The first domains is called TMaze (Bakker 2002), depicted in Figure 3.3, with a size of 10, which was initially proposed to test the



(a) Multiplicative for $\tau = 6$ and seed=67.



(b) Additive for $\tau = 6$ and seed=62.

Figure 3.9: TSNE plots for multiplicative and additive RNNs for various number of training samples.

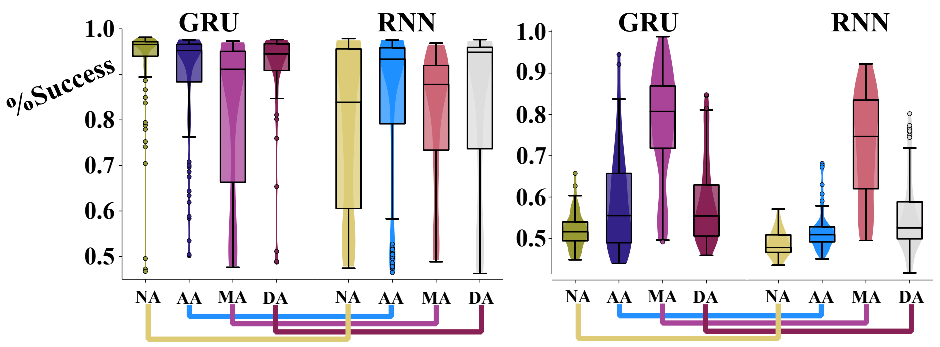


Figure 3.10: **(left)** Bakker's TMaze box plots and violin plots over the performance averaged over the final 10% with 50 independent runs. Trained over 300k steps with $\tau = 10$. All GRUs use a state size 6, while RNNs use a state size 20. The deep additive used an action encoding of $|\mathbf{a}| = 4$. **(right)** Directional TMaze comparison over the performance averaged over the final 10% of episodes with 100 independent runs trained over 300k steps with $\tau = 12$ for CELL (hidden size): RNN (30), AARNN (30), MARNN (18), DARNN (25, $|\mathbf{a}| = 15$), GRU (17), AAGRU (17), MAGRU (10), DAGRU (15, $|\mathbf{a}| = 8$).

capabilities of LSTMs in RL using Q-Learning. The environment is a long hallway with a T-junction at the end. The agent receives an observation indicating whether the goal state is in the north position or south position at the T-junction (which is randomly chosen at the start of the episode). The agent can take actions in the compass directions. On each step the agent receives a reward of -0.1 and in the final transition receives a reward of 4 or -1 depending if the agent was able to remember which direction the goal was in. The agent deterministically starts at the beginning of the hallway. The observation in the first state is $[1, 1, 0]$ if the goal state is located above the agent and $[0, 1, 1]$ if the goal state is below the agent. In the final state of the hallway the agent receives $[0, 1, 0]$ as an observation, and everywhere else the observation is $[1, 0, 1]$.

Our control agents are constructed similarly to those used in the Ring World environment. The agent's network is a single recurrent layer followed by a linear layer. A sweep is performed over the size of the hidden state and learning rates, and selected all variants of a cell type to have the same value. The network is trained over 300000 steps with further details reported in appendix A. The agent's performance is reported over the final 10% of episodes by averaging the agent success in reaching the correct goal. All results are reported using a box and whisker plot with the distribution. The upper and lower edges of the box represent the upper and lower quartiles respectively, with the median denoted by a line. The whiskers denote the maximum and minimum values, excluding outliers which are marked.

Shown in Figure 3.10 (left), all the cells have similar median performance with the GRU (with no action input) performing the best with the least amount of spread. This conclusion is the same across the size of the hidden state, where the multiplicative and factored variants performed poorly (see Appendix A for factored results). While this initially suggests the action embedding is not important beyond our simple Ring World experiment, notice the difference in how the environment's dynamics interact with the agent's action. In the TMaze, the underlying position of the agent is affected by only two of the actions (the East and West action), while the North and South actions only transition to a different state at the very end of the maze. Also, the agent's actions do not affect needs to be remembered, no matter what trajectory the agent sees the meaning of the first observation is always the same. Thus, these results are much less surprising. For example, the multiplicative variants will have to learn the update dynamics multiple times for the North and South actions.

To better replicate these dynamics in TMaze, a direction component is added to the underlying state. For example, many robotics systems must be able to orient and turn to progress in a maze, which I hypothesize actions will be critical for modeling the state. The agent can take an action moving forward, turning clockwise, or turning counter-clockwise. Instead of the observations only being a function of the position, the agents direction plays a critical role. In the first state, the agent receives the goal observation $[1, 1, 0]$ when facing the wall corresponding to the goal's direction. All other walls have the observation $[0, 1, 0]$, and when not facing a wall the agent receives the observation $[0, 0, 1]$. In DirectionalTMaze the agent is forced to contextualize its observation by the action it takes before or after seeing the observation. The state updates are evaluated using the same methodology as in the TMaze with results reported

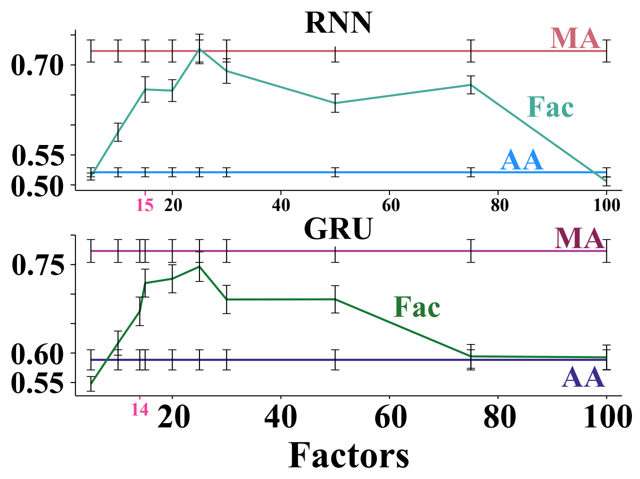


Figure 3.11: Sensitivity curves over number of factors M with standard error for the **(top)** FacRNN (30) and **(bottom)** FacGRU (17). All agents were trained over 300k steps. The data was generated by a sweep over the learning rate with 40 runs and compare to the data in figure 3.10. The red labels on the x-axis indicate when the network has the same number of parameters as the multiplicative.

in Figure 3.10 (right).

Now that the agent must be mindful of its orientation, the action again becomes a critical component in learning. The multiplicative variants outperforming all other variants in this domain. Without action, the GRU and RNN are unable to learn, and even the additive and deep additive versions are unable to learn in 300000 steps. Figure 3.11 reports the performance of a sweep over the number of factors and report the as compared to the multiplicative and additive variants. As the factors increase, generally the performance increases as well. This matches our expectations, as with increased factors the factored variants should better approximate the multiplicative variances. But there is a tradeoff when adding too many factors, causing performance to decrease substantially. While the factored variant has some interesting properties, the remaining experiments focus on the base architectures (NA, MA, AA, DA) and report full results with the factored variant in Appendix A.

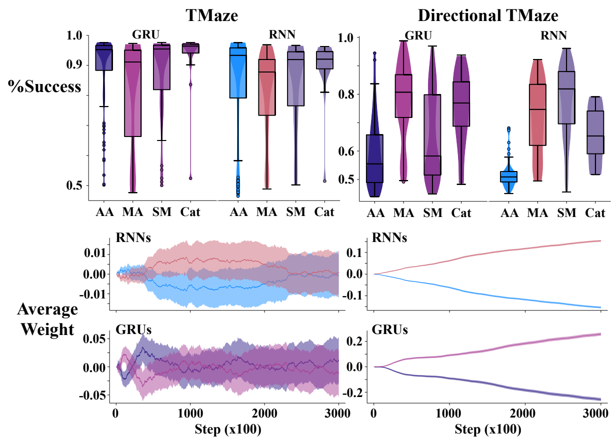


Figure 3.12: Two variants of combining cells. State size chosen based on procedures of previous environments. **(top)** Performance of success rates **(left)** TMaze with same basic parameters as above for CELL (hidden size): Softmax GRU (6), Cat GRU (6), Softmax RNN (20), Cat RNN (20). **(right)** Directional TMaze with same parameters as above for CELL (hidden size): Softmax GRU (8), Cat GRU (12), Softmax RNN (15), Cat RNN (22). **(bottom)** Average softmax weights of cells over training with standard error over runs.

3.7 Combining Cell Architectures

This section empirically evaluates combinations of the additive and multiplicative cells. These architectures are a minor step toward building an architecture which learns the structural bias currently hand designed.

The hidden state between an additive and multiplicative operation are combined through two techniques. The first is through an element-wise softmax. Both the additive and multiplicative have the same size hidden state ($\mathbf{s}^a$ and $\mathbf{s}^m$ respectively), and each element of the hidden states are weighted by

$$\mathbf{s}_i = \frac{e^{\theta_i^a}\mathbf{s}_i^a + e^{\theta_i^m}\mathbf{s}_i^m}{e^{\theta_i^a} + e^{\theta_i^m}}$$

where $\boldsymbol{\theta}^a, \boldsymbol{\theta}^m \in \mathbb{R}^n$. This should learn which cell to use depending on the structure of the problem. The second combination is through concatenating the two hidden state together $\mathbf{s} = cat(\mathbf{s}^a, \mathbf{s}^m)$. This gives more room for experts to add more state to the different architectures, but in this work the two architectures are fixed to have the same state size.

These combinations are compared to the original architectures in TMaze and Directional TMaze following the same procedure as above. These cells should perform as well as either the additive or the multiplicative (which ever is doing the best in the specific domain). The results can be seen in Figure 3.12. Overall, the softmax combination performs similarly or slightly better than the multiplicative version except in the Directional TMaze for the GRUs. In TMaze, concatenating the two states together performed better than the additive and multiplicative cells, but this operation worked slightly worse than the multiplicative in the Directional TMaze. To test the hypothesis that the softmax weighting should emphasize the better cell in a given domain the softmax weighting over the training period is reported. For TMaze the weightings end being approximately equivalent while the Directional TMaze shows a very distinct separation where the multiplicative is weighted significantly more and the additive is continually down-weighted.

3.8 Learning State Representations from Pixels

Finally, an empirical study in two environments with non-binary observations is performed. I am particularly interested in whether the recurrent architectures perform comparably when the observation needs to be transformed by fully connected layers, or when the observation is an image. These experiments only use the GRU cells. Full details can be found in Appendix A.4.

The first domain considered is a version of DirectionalTMaze which uses images instead of bit observations. The agent receives a gray scale image observation on every step of size 28×28. The agent sees a fully black screen when looking down the hallway, and a half white half black screen when looking at a wall. The agent observes an even (or odd) number sampled from the MNIST (LeCun et al. 2010) dataset when facing the direction of (or opposite of) the goal. The rewards are -1 on every step and 4 or -4 for entering the correct and incorrect goal position respectively. The same error metrics as in the prior TMaze environments are reported, with the environment's hallway being of length 6. Notice the hallway size is smaller and the negative reward is larger, this was to speed up learning for all architectures.

Results for the Image DirectionalTMaze can be seen in Figure 3.13. In this

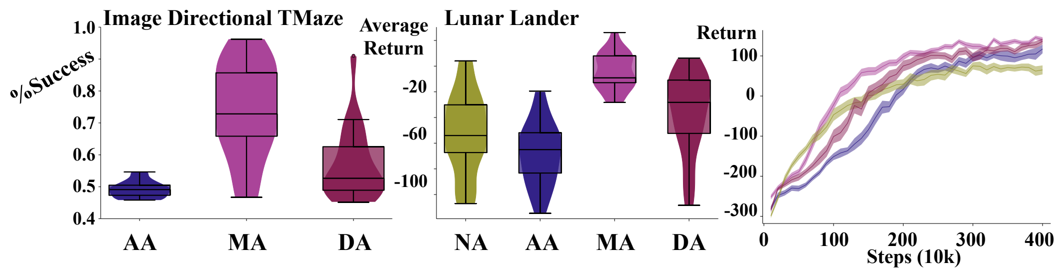


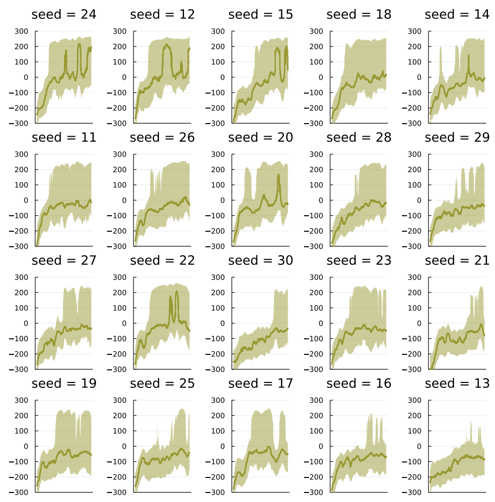
Figure 3.13: **(left)** Image Directional TMaze percent success over the final 10% of episodes for 20 runs for CELL (hidden size): AAGRU (70), MAGRU (32), DAGRU (45, $|\mathbf{a}| = 128$). Using ADAM trained over 400k steps, ($\tau = 20$). GRU omitted due to prior performance. **(center)** Lunar Lander average reward over all episodes for CELL (hidden size): GRU (154), AAGRU (152), MAGRU (64), DAGRU (152, $|\mathbf{a}| = 64$) and ($\tau = 16$). **(right)** Lunar Lander learning curves over total reward. Ribbons show standard error and a window averaging over 100k steps was used. Lunar Lander agents were trained for 20 independent runs for 4M steps.

domains, the multiplicative performs quite well, although not as well as in the simple version. The AAGRU is unable to learn in this setting, and the deep additive variant performs slightly better than the additive.

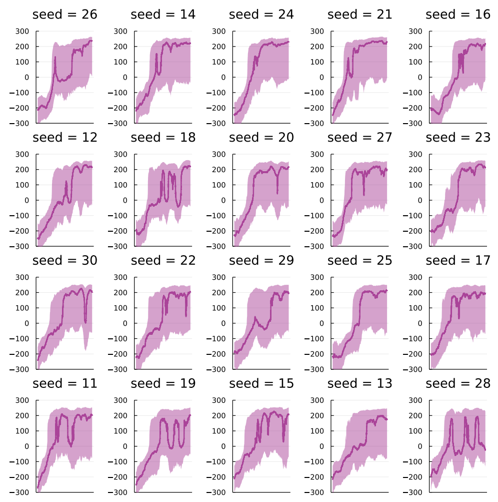
3.9 Learning State Representations from Agent-Centric Sensors

The second domain is a partially observable version of the LunarLander-v2 environment from OpenAI Gym (Brockman et al. 2016). The goal is to land a lander on the moon within a landing area. Further details and results can be found in Appendix A.4. The observation is modified by removing the anglular speed, and filtering the angle θ such that it is 1 if $-7.5 \leq \theta \leq 7.5$ and 0 otherwise. The average reward obtained over all episodes and learning curves are reported.

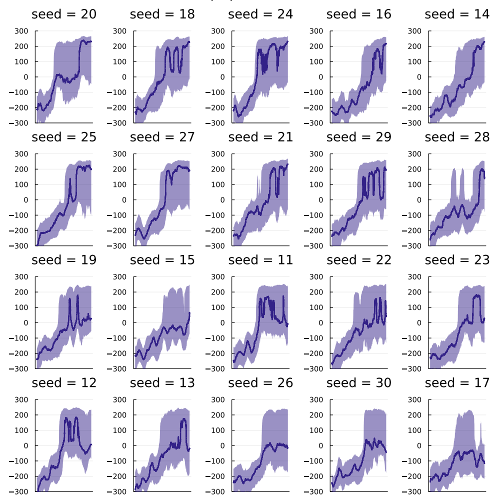
As seen in Figure 3.13, our findings generalize to this domain as well. The multiplicative variant improves over the factored (see Appendix A), additive, and deep additive variants significantly. In the LunarLander environment the multiplicative learns faster, reaching a policy which receives on average 100 total reward per episode. Both the additive and factored eventually learn



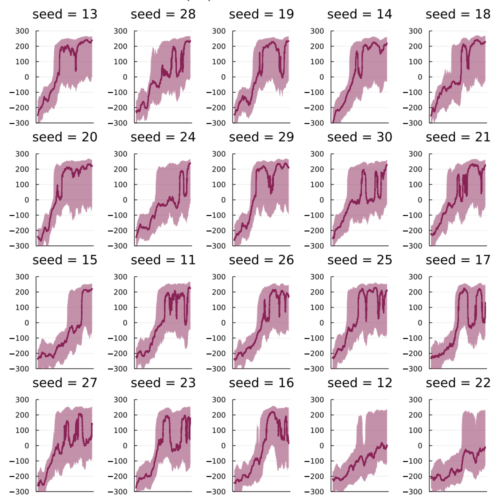
(a) GRU



(b) MAGRU



(c) AAGRU



(d) DAAGRU

Figure 3.14: Individual learning curves. Line is the median over 1000 episodes, with the shaded region as the 1st and 3rd quantile over the same window.

similar policies, while the standard GRU seems to perform less well (although not statistically significant from the additive variant). The average return is ~100 less than some of the best agents on this domains. The agent does this well 50% of the time, as seen in the individual median curves in Figure 3.14. This difference can be explained by the failure start states being more frequent than in the fully observed case.

3.10 Summary and Conclusions

This chapter empirically evaluated several strategies for incorporating the previous action into the state update of a recurrent neural network. The impact of this choice was shown to have a large impact on an RL agent's performance in several environments from several observation types. These empirical results suggest that the multiplicative operation performs the best even when using a smaller state vector, and the factored and the deep additive versions perform marginally better than the additive versions in most domains.

While the multiplicative seems to be the clear winner on the tested domains, it is important to note not all domains require this architecture. One interesting strategy could be to use the softmax combined cells to decide which cell to use in your final architecture (by looking at the softmax weighting). One could also imagine an architecture which is able to learn which cell to use conditioned on the history of the agent (see Section 10.2.2). Until better architectures for RL are defined this choice is left to system designers. While the additive and deep additive versions under-performed compared to the other encodings, it still out-performed naively using RNNs without action input.

What is apparent in these experiments and other empirical evidence recently gathered on the performance of recurrent architectures in the online setting (Schlegel, Jacobsen, et al. 2021; Rafiee, Abbas, et al. 2022) is that the methods and architectures developed and utilized by supervised learning might not be suitable for the reinforcement learning problem. This paper uncovered a simple choice can have a large impact, and provides some evidence that the assumptions made in supervised learning might be holding back recurrent architectures in the reinforcement learning setting (see Section 10.2.1 for details). Small, focused studies using recurrent agents in controlled experiments will continue to produce insights on the limitations of the base algorithms and continue to inspire future algorithm developments.

Chapter 4

General Value Function Networks

In this chapter, a predictive representations of state approach known as general value function networks is introduced. In short, these networks constrain the state of a recurrent network to answer predictive questions in the form of GVFs. This chapter defines the core framework of the general value function network (GVFN). The key contributions include the restriction of predictive questions to be composed in acyclic graphs, and extensions to include the set of predictions made by PSRs, TDNets, and other forecasting networks. In later chapters a set of learning algorithms for these architectures are derived (Chapter 5), and then empirically evaluated (Chapter 6).

4.1 Representations of State

Most domains of interest are partially observable, where an agent only observes a limited part of the state. In such a setting, if the agent uses only the immediate observations, then it has insufficient information to make accurate predictions or decisions. A natural approach to overcome partial observability is for the agent to maintain a history of its interaction with the world. For example, consider an agent in a large and empty room with low-powered sensors that reach only a few meters. In the middle of the room, with just the immediate sensor readings, the agent cannot know how far it is from a wall. Once the agent reaches a wall, though, it can determine its distance from the wall in the future by remembering this interaction. This simple strategy, however, can be problematic if a long history length is needed (McCallum et al. 1996).

State construction enables the agent to overcome partial observability, with a more compact representation than an explicit history. Because most environments and datasets are partially observable—in time series prediction, in modeling dynamical systems and in reinforcement learning—there is a large literature on state construction. These strategies can be separated into Objective-state and Subjective-state approaches.

Objective-state approaches specify a true latent space, and use observations to identify this latent state. An objective representation is one that is defined in human-terms, external to the agent's data-stream of interaction. They typically require an expert to provide feature generators or models of the agent's motion and sensor apparatus. Many approaches are designed for a discrete set of latent states, including HMMs (Baum and Petrie 1966) and POMDPs (Kaelbling et al. 1998). A classical example is Simultaneous Localization and Mapping, where the agent attempts to extract its position and orientation as a part of the state (Durrant-Whyte and Bailey 2006). These methods are particularly useful in applications where the dynamics are well-understood or provided, and so accurate transitions can be used in the explicit models. When models need to be estimated or the latent space is unknown, however, these methods either cannot be applied or are prone to misspecification.

The goal of subjective-state approaches, on the other hand, is to construct an internal state only from a stream of experience. This contrasts objective-state approaches in two key ways. First, the agent is not provided with a true latent space to identify. Second, the agent need not identify a true latent state, even if there is one. Rather, it only needs to identify an internal state that is sufficient for making predictions about target variables of interest. Such a state will likely not correspond to objective quantities like meters and angles, but could be much simpler than the true latent state and can be readily learned from the data stream. Examples of subjective-state approaches to state construction include Recurrent Neural Networks (RNNs) (Hopfield 1982; Lin 1993), Predictive State Representations (PSRs) (Littman and Sutton 2002) and TD Networks (Sutton and Tanner 2005).

I have discussed several causes and remedies for the instability of RNNs. An alternative direction, that requires more domain expertise than RNN expertise, is to use predictions as auxiliary losses. Auxiliary unsupervised losses have been used in NLP to improve trainability (Trinh et al. 2018). Less directly, auxiliary losses were used in reinforcement learning (Jaderberg et al. 2017) and for modeling dynamical systems (Venkatraman et al. 2017), to improve the quality of the representation; this is a slightly different but nonetheless related goal to trainability. The use of predictions for auxiliary losses is an elegant way to constrain the RNN, because the system designers are likely to have some understanding of the relevant system components to predict. For the larger goals of AI, augmenting the RNN with additional predictions is promising because one could imagine the agent discovering these predictions autonomously—predictions by design are grounded in the data stream and learnable without human supervision. Nonetheless, the use of predictions as auxiliary tasks provides a more indirect (second-order) mechanism to influence the state variables. In this work, I ask: is there utility in directly constraining states to be predictions?

To answer this question, a practical approach for learning RNNs where the internal state corresponds to predictions is needed. I propose a new RNN architecture, where the hidden state is constrained to be multi-step predictions, using an explicit loss function on the hidden state. Specifically, the architecture uses general policy-contingent, multi-step predictions: General Value Functions (Sutton, Modayil, et al. 2011).

4.2 Constraining State to be Predictions

Let us start in a simpler setting and explain how the hidden units could be trained to be n-horizon predictions about the future. Imagine you have a multi-dimensional time series of a power-plant, consisting of d sensory observations with the first sensor corresponding to water temperature. Your goal is to make a hidden node in your RNN predict the water temperature in 10 steps, because you think this feature is useful to make other predictions about the future.

This can be done simply by adding the following loss: $(\mathbf{s}_{t,1} - \mathbf{x}_{t+10,1})^2$. The combined loss $L_t(\mathbf{W})$ on time-step t is

$$\mathcal{L}_t(\mathbf{W}) \overset{\text{def}}{=} \ell(\hat{y}_t, y_t) + (\mathbf{s}_{t,1} - \mathbf{x}_{t+10,1})^2 \tag{4.1}$$

where both $\hat{y}_t$ and $\mathbf{s}_t$ are implicitly functions of $\mathbf{W}$. This loss still encourages the RNN to find a hidden state $\mathbf{s}_t$ that predicts y_t well. There is likely a whole space of solutions that have similar accuracy for this prediction. The second loss constrains this search to pick a solution where the first state node is a prediction about an observation 10 steps into the future. This second term can be seen as a regularizer on the network, specifying a preference on the learned solution. In general, more than one state node—even all of $\mathbf{s}_t$—could be learned to be predictions about the future.

The difficulty in training such a state representation depends on the chosen targets. For example, long horizon targets—such as 100 steps rather than 10 steps into the future—can be high variance. Even if such a predictive feature could be useful, it may be difficult to learn accurately and could make the state-update less stable. Using n-horizon predictions also requires a delay in the update: the agent must wait 100 steps to see the target to update the state at time t.

Therefore, I propose to restrict ourselves to a class of prediction that have been shown to be more robust to these issues, GVFs (Sutton, Modayil, et al. 2011; Modayil et al. 2014; van Hasselt and Sutton 2015). Algorithms exist to estimate these predictions online, without having to wait to see outcomes in the future. This property of GVFs is *independence of span* (van Hasselt and Sutton 2015), meaning learning can be achieved with computation and memory independent of the horizon. Such a property is doubly critical for predictions within an RNN, as it is more likely these predictions can be estimated sufficiently quickly to be usable as state. Further, there is some evidence that this class of predictions is sufficient for a broad range of predictions about the future (Pezzulo 2008; Sutton, Modayil, et al. 2011; Modayil et al. 2014; A. White 2015; Banino et al. 2018; Momennejad and Howard 2018), and so the restriction to GVFs does not significantly limit representability.

4.2.1 The components of a predictive state approach

Every approach to constructing state with predictions has three core components. The first is how a predictive question is asked or phrased. This can have dramatic affects on the hypothesis/function class of the predictive state, and induce large differences in the underlying algorithmic assumptions used for training. The second is in how the questions will be answered. An approach must consider the base function classes used to represent answers, the abstractions (either temporally or otherwise), and the learning algorithms applied to the architecture. The third, and probably less studied, is that of discovery. Discovery is the automatic specification of predictive questions (e.g. GVFs in a GVFN).

Choosing the semantics of how predictive questions are asked will have major effects in the question's discoverability and answerability. PSRs use histories of action observation pairs to construct predictive question, where the answer is a representation of the probability of the sequence of observations being seen given a history and the agent follows the action sequence (Littman and Sutton 2002; Singh, James, et al. 2004). TDNs use an *question network* which is a graph of target dependencies with the core nodes representing specific parts of the observational space. This graph can be many layers, and is acyclic with the single exception of a self-recurrence denoting discounting. Both TDNs and PSRs were originally defined only using primitive actions to ask questions, but were later extended to included temporally abstract options through option-conditional TDNs (Sutton and Tanner 2005; Rafols, Koop, et al. 2006) and hierarchical PSRs (HPSRs) (Wolfe and Singh 2006).

The second component is that of learning and representing the answers of the predictive questions. While respectively different questions, they are deeply connected in the design of any system. The original work in PSRs restricted the sets of observations and actions to be finite. The reason this was needed was how the answers were represented, given a history and sequence of actions for the sequence of observations to not be trivially zero the observations must be sampled according to a mass function. This was addressed in later work using kernel density estimation and information-theoretic tools to realize PSRs in the case of continuous observations and actions (Wingate and Singh 2007). The answers were then represented as a vector of predicted values for the core tests, which could be updated incrementally with new observations. TDNs use artificial neural networks to underly their representation of answers. While the organization of nodes is not restricted (Sutton and Tanner 2005), most of the empirical results shown can be described as using a recurrent neural network. Schlegel, Jacobsen, et al. 2021 make this restriction more apparent, where they explicitly learn the predictive representation as the state of a recurrent network. This simplifies the comparison to non-predictive subjective state approaches (i.e. RNNs), while also enabling the application of backpropagation through time and real-time recurrent learning.

The third and final component is that of discovery. Discovery is the automatic specification of predictive questions to use in learning the predictive state. PSRs approached discovery by exploring the set of tests to construct a core set that enables all other tests to be answered (James et al. 2005; Wolfe, James, et al. 2005; Mccracken and Bowling 2006; Wingate and Singh 2007). This objective is trying to find a sufficient statistic of the history for all predictions, and has been discussed in various forms (Subramanian et al. 2022). I conjecture that finding such a state is not feasible in large complex problems, and searching for such a state would be a poor use of a finite set of computational resources. Instead, the agent should focus on finding a set of questions which is useful for the agents overarching goals—for example, maximizing the return in the control problem. Along this new objective several other approaches have been proposed. Generate and test is a general algorithm for searching through a large space with opaque dynamics (Mahmood and Sutton 2013; Javed 2020). Another approach is to define the predictive questions as a parametric optimization problem and use meta-gradient descent (Bacon and Precup 2017; Veeriah et al. 2019). This approach splits the problem into two optimization problems: an inner problem and an outer problem. The inner optimization consists of the usual control or prediction procedure, where the agent seeks to maximize the discounted return or lower prediction error. The outer optimization calculates gradients through this procedure, with respect to the meta-parameters.

Given a predictive approach to state building requires consideration of these difficult algorithmic choices, a natural question arises “Why shouldn't we use non-predictive subjective based approaches for learning state, such as the usual recurrent networks?”. While this thesis won't provide (or seek) a conclusive answer to this question, predictive approaches to state construction may have a positive effect on a system's ability to generalize and learn a state representation. This is stated in the *Predictive Representation Hypothesis* (Schaul and Ring 2013):

> a(n) *(explicit) predictive representation of state* will be able to continually construct useful generalizations of the regularities in an environment.

An *(explicit) predictive representation of state* is an algorithm, or architecture, which constrains the state to be predictions which minimize an objective separate (or jointly) from the agent's general goal in an environment. This class of algorithms includes PSRs, TDNs, GVFNs, and several others. Because the state will be made of small-specific predictive questions of the agent's sensory-motor stream, as the distributions of the underlying dynamics shift the answers to the questions should appropriately shift as well.

4.3 GVF Networks

In this section, GVF Networks are introduced. The GVF Network (GVFN) is an RNN architecture where hidden states are constrained to predict policy-contingent, multi-step outcomes about the future. I first describe GVFs and the GVF Network (GVFN) architecture. Several related predictive approaches, such as TD Networks, are discussed in Section 4.5, after introducing GVFNs.

First, the definition of GVFs (Sutton, Modayil, et al. 2011) needs to be expanded to the partially observable setting, to use them within RNNs. The first step is to replace state with histories. Remember in Section 2.4 we introduced the set of minimal histories that enables the Markov property for the distribution over next observation

$$\mathcal{H} = \left\{ \mathbf{h}_k = (\mathbf{o}_0, a_0, \dots, \mathbf{o}_k) \mid {\scriptstyle \text{(Markov property)} p(\mathbf{o}_{k+1}|\mathbf{h}_k, a_k) = p(\mathbf{o}_{k+1}|\mathbf{o}_{-1}, a_{-1}, \mathbf{h}_k a_k), \atop \text{(Minimal history) } p(\mathbf{o}_{k+1}|\mathbf{h}_k) \neq p(\mathbf{o}_{k+1}|\mathbf{o}_1, a_1, \dots, a_{k-1}, \mathbf{o}_k)} \right\} \tag{4.2}$$

As defined in Section 2.6, a GVF question is a tuple (π, C, γ) composed of a policy $\pi : \mathcal{H} \times \mathcal{A} \to [0, \infty)$, cumulant $C : \mathcal{H} \times \mathcal{A} \times \mathcal{H} \to \mathbb{R}$ and continuation (or discount) function $\gamma : \mathcal{H} \times \mathcal{A} \times \mathcal{H} \to [0, 1]$. On time step t, the agent is in H_t, takes actions A_t, transitions to H_{t+1} and observes cumulant C_{t+1} and continuation γ_{t+1}. The answer to a GVF question is defined as the value function, $V : \mathcal{H} \to \mathbb{R}$, which gives the expected, cumulative discounted cumulant from any history $\mathbf{h}_t \in \mathcal{H}$. The value function which can be defined recursively with a Bellman equation as

$$\begin{aligned} V(\mathbf{h}_t) &\overset{\text{def}}{=} \mathbb{E}[C_{t+1} + \gamma_{t+1} V(H_{t+1}) \mid H_t = \mathbf{h}_t, A_t \sim \pi(\cdot|\mathbf{h}_t)] \qquad (4.3) \\ &= \sum_{a_t \in \mathcal{A}} \pi(a_t|\mathbf{h}_t) \sum_{\mathbf{h}_{t+1} \in \mathcal{H}} p(\mathbf{h}_{t+1}|\mathbf{h}_t, a_t)[C(\mathbf{h}_t, a_t, \mathbf{h}_{t+1}) + \\ &\qquad \gamma(\mathbf{h}_t, a_t, \mathbf{h}_{t+1}) V(\mathbf{h}_{t+1})]. \end{aligned}$$

The sums can be replaced with integrals if $\mathcal{A}$ or $\mathcal{O}$ are continuous sets. The definitions and theory associated with the GVFN assumes that $\mathcal{H}$ is a finite set, for simplicity; however, the theory can be extended to infinite and uncountable sets.

A GVFN is an RNN, and so is a state-update function f, but with the additional criteria that each element in $\mathbf{s}_t$ corresponds to a prediction—to a GVF. A GVFN is composed of N GVFs, with each hidden state component $\mathbf{s}_{t,j}$ trained such that at time-step t, $\mathbf{s}_{t,j} \approx V^{(j)}(\mathbf{s}_t)$ for the jth GVF and history $\mathbf{s}_t$. Each hidden state component, therefore, is a prediction about a multi-step policy-contingent question. The hidden state is updated recurrently as $\mathbf{s}_t \overset{\text{def}}{=} f_{\mathbf{W}}(\mathbf{s}_{t-1}, \mathbf{x}_t)$ for a parametrized function $f_{\mathbf{W}}$, where $\mathbf{x}_t = [a_{t-1}, \mathbf{o}_t]$ and $f_{\mathbf{W}}$ is trained so that $\mathbf{s}_j \approx V^{(j)}(\mathbf{h}_t)$. This is summarized in Figure 4.1.

General value functions provide a rich language for encoding predictive knowledge. In their simplest form, GVFs with constant γ correspond to multi-timescale predictions referred to as Nexting predictions (Modayil et al. 2014).

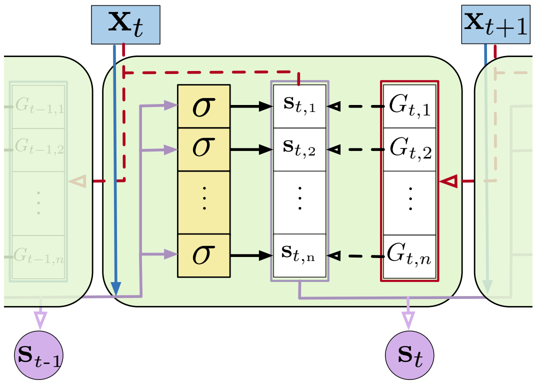


Figure 4.1: GVF Networks (GVFNs), where each state component $\mathbf{s}_{t,i}$ is updated towards the return $G_{t,i} \overset{\text{def}}{=} C_{t+1}^{(i)} + \gamma_{t+1}^{(i)} \mathbf{s}_{t+1,i}$ for the ith GVF. The solid forward arrows indicate how state is updated; in fact, the update is the same as a standard RNN. The difference is with the dotted lines, that indicate training. The dotted black arrows indicate the targets for the state components. The dotted red arrows indicate that the target $G_{t,i}$ are created using the observation and state on the next step.

Allowing γ to change as a function of state or history, GVF predictions can combine finite-horizon prediction with predictions that terminate when specific outcomes are observed (Modayil et al. 2014).

To build some intuition, below are some examples in Compass World. This environment is used in our experiments and depicted in Figure 4.2. Compass World is a grid world where the agent is only provided information about the color directly in front it. This world is partially observable, with all the tiles in the middle having a white observation, with the only distinguishing color information available to the agent at the walls. The actions taken by the agent are to move forward, turn left, or turn right.

In this environment, the agent might want to know if it is facing the red wall. This can be specified as a GVF question: “If I go forward until I hit a wall, what is the probability I will see red?”. The policy is to always go forward. If the current observation is ‘Red’, then the cumulant is 1; otherwise it is zero. The continuation γ is 1 everywhere, except when the agent hits a wall and sees a color; then it becomes zero. The sampled return from a state is 1.0 if the agent is facing the Red wall, because going forward will result in summing many zeros plus a 1 right before termination. If the agent is not facing the

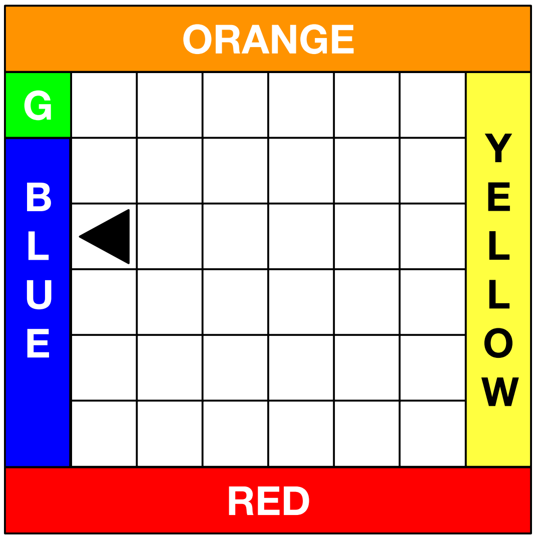


Figure 4.2: The Compass World: A partially observable grid world with observations of the color directly in front of the agent. **Actions:** The agent can take the actions Move Forward (one cell), Turn Left, and Turn Right. **Observations:** The agent observes the color of the grid cell it is facing. This means the agent can only observe a color if it is at the wall and facing outwards. The agent depicted as an arrow would see Blue. In the middle of the world, the agent sees White. **Goal:** The agent's goal is to make accurate predictions about which direction it is facing.

Red wall, the return is 0, because the agent terminates when hitting the wall but only sees cumulants that are zero for the entire trajectory. Because the outcome is deterministic, the probabilities are 1 or 0.

The agent could also ask about how frequently it will see Red, within a horizon of about 10 steps. An approximation to this question can be obtained by using a constant continuation of $\gamma = 0.9$. The intuition for this comes from thinking of $1 - \gamma$ as a success probability for a geometric distribution: the probability of successfully terminating. The mean of this geometric distribution is $\frac{1}{1-\gamma}$—which in this case is $\frac{1}{1-0.9} = 10$—provides the expected number of steps until the first success. Recall that termination indicates that a return is cut-off, and so a cumulant is not included in the sum after termination. This probabilistic termination means that even if Red is seen after 10 steps, it will still be included in the return. However, it does indicate its contribution has been significantly decayed. This exponential prediction loses precision, and so the GVF only provides an approximation to this question.

The agent could also also ask if it will see Red, within a horizon of about 10 steps. In this case, the continuation would be 0.9 until the agent observed Red, at which point it would become zero (indicating termination). The GVF answer corresponds to a discounted probability of observing Red, with a smaller number if Red is observed further in the future. If the agent always see Red in 1 step from $\mathbf{h}_t$, then it observes $C_{t+1} = 1$ and $\gamma_{t+1} = 0$ and the value is precisely 1. If the agent sees Red in 2 steps from $\mathbf{h}_t$, then $C_{t+1} = 0, \gamma_{t+1} = 0.9, C_{t+2} = 1$ and $\gamma_{t+2} = 0$ resulting in a value of 0.9. If the agent sees Red in 10 steps from $\mathbf{h}_t$, then the value is $0.9^9 \approx 0.4$. If just a few more steps into the future, say 15 steps, then the value would be 0.2. The magnitudes start to get quite low, indicating that it is less likely to observe Red in this window.

Notice that though the cumulants and continuation functions are defined on the underlying (unknown) state $\mathbf{h}_t$, this is a generalization of defining it on the observations. The observations are a function of state; the cumulants and continuations γ that are defined on observations are therefore defined on $\mathbf{h}_t$. In the examples above, these functions were defined using just the observations. More generally, these functions are a part of the problem definition. This means they could be defined using short histories, or other separate summaries of the history. As discussed in Section 5, one can also consider cumulants that are a function of our own predictions or constructed state.

A natural question is how these GVFs are chosen. This problem corresponds to the discovery problem for predictive representations. This chapter focuses on the utility of this architecture with simple heuristics or expert chosen GVFs. I briefly discuss simple ideas for discovery in Chapter 8, but leave a more systematic investigation of the discovery problem to future work.

4.4 A Case Study using GVFNs for Time Series Prediction

GVFNs can be used for time series prediction by simply assuming that a fixed (unknown) policy generates the data. The GVFs within the network are assumed to have this same fixed unknown policy in common, but differ

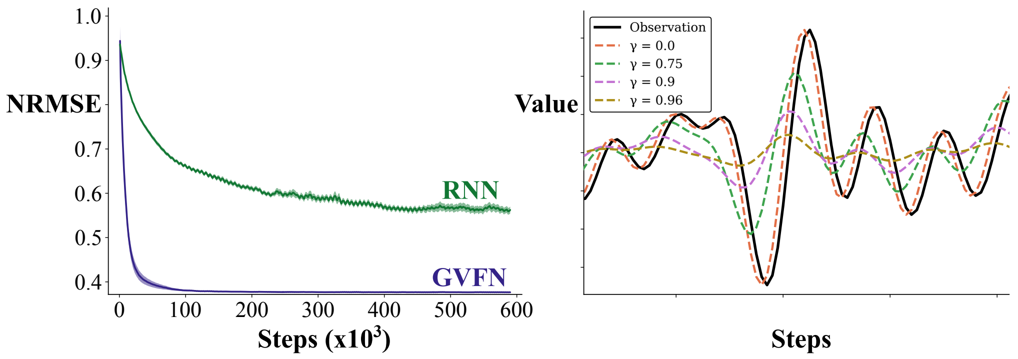


Figure 4.3: **(left)** Example learning curve for MSO with GVFNs and simple RNNs **(right)** The returns for different γ, corresponding to GVFs in the GVFN, for a small section of the MSO time series dataset. The dotted red line for $\gamma = 0$ looks overlayed with the time series plotted in black, but is actually the observation one step in the future.

in the pair of continuation and cumulant functions. For a multi-variate time series, one GVF could have a cumulant corresponding to the first entry of the observation on the next time step, and another GVF could use the second entry. Even for a single-variate time series, meaningfully different cumulants be defined. For example, one GVF could correspond to the probability that the observation becomes larger than 1. The cumulant would be zero until this event occurs, at which point it would be 1. In Figure 4.3 (left) preliminary results using a GVFN to forecast 12-steps into the future on the single-variate Multiple Super-imposed Oscillator (MSO) time series dataset. The full empirical set-up is discussed in Section 6.1, and here some insights relevant to building intuition for how to use GVFNs are provided.

The GVFN consists of a recurrent, constrained layer of 128 GVFS with γs spaced linearly in $[0.1, 0.97]$ to learn the state. To make predictions, feedforward layers from this recurrent layer can be added; here a ReLu layer is added for additional nonlinearity in the prediction. For comparison, a simple RNN is also included, which similarily uses an additional ReLu layer after its recurrent layer. The prediction target is the observation 12 steps into the future. Both the RNN and GVFN have to wait 12 steps to see the accuracy of their prediction, delaying updates based on the target by 12 steps. The GVFN, however, can use the loss on the state at each step, and so more directly influence the value of states with the most recent observations. Both methods use p-BPTT, with truncation p. With a sufficiently high p, both perform well (see Section 6.1 for results with many p). The result reported here is for $p = 1$, where the GVFN already obtains near-optimal performance.

It might be surprising that this simple GVFN, with GVFs only differing in continuation γ, can perform well. For time series data, however, such constant γ predictions provide anticipatory information about observations in the future. To see why, the time series as well as returns for $\gamma \in \{0, 0.75, 0.9, 0.96\}$ as dotted lines are shown in Figure 4.3 (right). These returns reflect the type of information that would be provided by a GVF prediction. At each time point t on the x-axis, the smaller γ, like $\gamma = 0.75$ as dotted green, anticipate the observations in a nearby window. If the time series is starting to rise in the near future, then the dotted green starts to rise right now. Returns can thus provide useful predictive information about increases and decreases that are expected to soon appear in the time series. Notice that the magnitude of the returns are approximately equal. For practical use, the magnitude of each GVF prediction should be similar, to avoid large differences in magnitude between state variables. With large γ, however, the return becomes large and so too does the value function. The standard fix to this is straightforward: each GVF uses a scaled cumulant of $(1 - \gamma)o_{t+1}$.

Notice, though, that there is a trade-off between anticipating a cumulant farther into the future and the precision of predictions about the future. Returns with lower continuations predict trends closer to when they occur in the dataset and have higher resolution. Returns with higher continuations anticipate changes further in the future, at the cost of smoothing over the detailed changes in the dataset. By using both lower and higher continuations, the benefits of both can be obtained. This simple heuristic—GVFs with the same cumulant and varying γ—as a general purpose heuristic is further discussed in Chapter 8.

4.5 Connections to Other Predictive State Approaches

The idea that an agent's knowledge might be represented as predictions has a long history in machine learning. The references to such a predictive approach can be found in the work of Becker (1973), Drescher (1991), and Cunningham (2013) who hypothesized that agents would construct their understanding of the world from interaction, rather than human engineering. These ideas inspired work on predictive state representations (PSRs) (Littman and Sutton 2002), as an approach to modeling dynamical systems. Simply put, a PSR can predict all possible interactions between an agent and it's environment by reweighting a minimal collection of core test (sequence of actions and observations) and their predictions, without the need for a finite history or dynamics model. Extensions to high-dimensional continuous tasks have demonstrated that the predictive approach to dynamical system modeling is competitive with state-of-the-art system identification methods (Hsu et al. 2012). PSRs can be combined with options (Wolfe and Singh 2006), and some work suggests discovery of the core tests is possible (Mccracken and Bowling 2006). Work in closing the learning-planning loop using predictive state representations lead to a spectral algorithm for learning a PSR directly from action-observation sequences (Boots et al. 2011).

TD networks (Sutton and Tanner 2005) were introduced after PSRs, and inspired by the PSR approach to state construction that is grounded in observations. GVFNs build on and are a strict generalization of TD networks. A TD network (Sutton and Tanner 2005) is similarly composed of N predictions, and updates using the current observation and previous step predictions like an RNN. TD networks with options (Rafols, Ring, et al. 2005) condition the predictions on temporally extended actions similar to GVF Networks, but do not incorporate several of the recent modernizations around GVFs, including state-dependent discounting and convergent off-policy training methods. The key differences, then, between GVF Networks and TD networks is in how the question networks are expressed and subsequently how they can be answered.

GVF Networks are less cumbersome to specify, because they use the language of GVFs. Further, once in this language, it is more straightforward to apply algorithms designed for learning GVFs.

More recently, there has been an effort to combine the benefits of PSRs and RNNs. This began with work on Predictive State Inference Machines (PSIMs) (Sun et al. 2016), for inference in linear dynamical systems. The state is learned in a supervised way, by using statistics of the future k observations as targets for the predictive state. This earlier work focused on inference in linear dynamical systems, and did not state a clear connection to RNNs. Later work more explicitly combines PSRs and RNNs (Downey et al. 2017; Choromanski et al. 2018), but restricts the RNN architecture to a bilinear update to encode the PSR update for predictive state. In parallel, another strategy to incorporate ideas from PSRs into RNNs, without restricting the RNN architecture, called Predictive State Decoders (PSDs) (Venkatraman et al. 2017). Instead of constraining internal state to be predictions about future observations, statistics about future observations are used as auxiliary tasks in the RNN.

Of all these approaches, the most directly related to GVFNs is PSIMs. This connection is most clear from the PSIM objective (Sun et al. 2016, Equation 8), where the goal is to make predictive state match a vector of statistics about future outcomes. There are some key differences, mainly due to a focus on offline estimation in PSIMs. The predictive questions in PSIMs are typically about observations 1-step, 2-step up to k-steps into the future. To use such targets, batches of data need to be gathered and statistics computed offline to create the targets. Further, the state-update (filtering) function is trained using an alternating minimization strategy, with an algorithm called DAgger, rather than with algorithms for RNNs. Nonetheless, the motivation is similar: using an explicit objective to encourage internal state to be a predictive state.

A natural question, then, is whether the types of questions used by GVFNs provides advantages over PSIMs. Unlike k-step predictions in the future, GVFs allow questions about outcomes infinitely into the future, through the use of cumulative discounted sums. Such predictions, though, do not provide high precision about such future events. As motivated in Section 4.2, GVFs should be easier to learn online. In our experiments, a baseline, called a Forecast Network, that uses k-step predictions as predictive features, is included to provide some evidence that GVFs are more suitable as predictive features for online agents.

Researchers in reinforcement learning, decision making, and artificial intelligence are not alone in asking if decision making systems use predictions to effectively navigate their world (Hawkins and Blakeslee 2004; Bubic et al. 2010; Clark 2013). Anticipation (Butz et al. 2003; Pezzulo et al. 2008)—which has similar properties to the GVF approach to prediction—has been used to mean elevated processing prior to an event (also prediction) as well as the overall effect of prediction on an agent behavior. An agent can anticipate an event in the future, and act accordingly. This requires the agent's policy to be defined in terms of predictions, or for the representation to have predictive/anticipatory properties. Hierarchical predictive coding (Rao and Ballard 1999; Huang and Rao 2011) was used to explain non-classical interference observed in the visual cortex. In this approach, feedback connections transport predictions (or priors) from higher layers to lower layers to give context to the current observations. Prospective codes (Schütz-Bosbach and Prinz 2007) take the theory of prospection and encode future events as representations used for planning and simulation. While there is evidence to suggest organic decision making systems are directed forward in their representation of the world, memory and "postdiction" both play an important, separate role in building a systems underlying representations (Soga et al. 2009; Synofzik et al. 2013). While I focus on two distinct classes in this thesis (i.e. predictive and postdictive), future architectures should be built to take advantage of both approaches.

While GVFs provide, in our opinion, a better language to ask complex predictive questions, one should understand what is given up when moving away from PSRs and PSIMs in predictive power. In the next two sections it is shown that the questions used in PSIMs and PSRs can be posed by GVFs not only through standard cumulants, but also through composite GVFs.

Below I use composite GVFs to show the potential predictive power of complex networks of GVFs. See Chapter 7 for a more detailed exploration into some of the properties of composite predictions.

4.5.1 k-step Forecasts using GVFs

I first show how to construct k-step forecasts using GVFs and then move to the more complex case of the core-tests in PSRs. The simplest way to make k-step forecasts with a GVF is to construct a cumulant function that requires the agent to receive k new observations and then update the forecast through any usual means. While this is reasonable in the offline setting, when training online our agent should be able to update predictions as soon as possible without waiting for k time-steps. This can be done trough composite predictions. With myopic discounts (i.e. $\gamma = 0$) a chain of GVFs can be constructed such that each GVF in the chain has a cumulant of the prior GVF's prediction on the next time-step. The $(k-1)th$ GVF in this sequence will be predicting the observation of the first GVF k steps into the future.

4.5.2 Core-tests in Predictive State Representations can be Defined by Composite GVFs

A predictive state representation is made up of a finite set of core-tests $\{q_1, q_2, \ldots, q_n\}$ and a set of likelihoods indicating the probability of seeing the core-test given the current history $p(q_1|H_t)$. The history is constructed as a sequence of observation-actions from the beginning of the agent's lifetime $\mathcal{H}_t = \{o_0 a_0 o_1 a_1 \ldots o_t\}$. Each core-test is a sequence $q = \{a_0, o_1, a_1, o_2, \ldots, a_{l-1}, o_l\}$. Note how the sequence starts with the pair (a_0, o_1), which is different from the usual notation used in the PSR literature.

To construct a single core-test using composite GVFs a chain of myopic predictions much like the k-step forecasts can be used. The first GVF in the chain of GVFs will have a cumulant $c(o_t, a_t, o_{t+1}) = \mathbb{1}(a_t = a_{l-1}, o_{t+1} = o_l)$, where ⊮ is the indicator function. Given a history $\mathcal{H}_t$, the expected target of this GVF will be $v^l = \mathbb{E}[\mathbb{1}(a_t = a_{l-1}, o_{t+1} = o_l)|\mathcal{H}_t] = p(a_{l-1}, o_l|\mathcal{H}_t)$. Now if GVFs are composed together, where subsequent GVFs are chained according to the cumulant function $c(o_t, a_t, o_{t+1}, \hat{v}_{l-i}) = \mathbb{1}(a_t = a_{l-i-1}, o_{t+1} = o_{l-i})$ and $\hat{v}^{l-i}_{t+1}$ (where the "and" represents a product). As the chain continues the final GVF should represent the likelihood $p(q|\mathcal{H}_t)$ which is the likelihood for a PSR test. I prove the length two case in the following lemma, but the more general case very easily extends from this case.

Lemma 4.5.1. *Given a length 2 sequence of actions and observations $\mathcal{S} = a_0, o_1, a_1, o_2$ and a history $\mathcal{H}_t$, the probability of seeing the sequence $\mathcal{S}$ given the history can be represented as two composed value function targets.*

Proof. The above corollary can be easily proven through the rules of conditional expectation and probabilities. The probability of seeing a sequence $\mathcal{S}$ given a history $\mathcal{H}_t$ can be written as

$$p(A_t = a_0, O_{t+1} = o_1, A_{t+1} = a_1, O_{t+2} = o_2|\mathcal{H}_t) = \\ \mathbb{E}[\mathbb{1}\{A_t = a_0, O_{t+1} = o_1, A_{t+1} = a_1, O_{t+2} = o_2|\}\mathcal{H}_t]$$

where $\mathbb{1}$ is the indicator function. Now following the procedure laid out above to build composite GVFs. The first GVF in the chain will have a cumulant $c_2(o_t, a_t, o_{t+1}) = \mathbb{1}\{A_t = a_1, O_{t+1} = o_2\}$ with a myopic discount (i.e. $\gamma = 0$). The resulting value function prediction is

$$V_2(\mathcal{H}_t) = \mathbb{E}[\mathbb{1}(a_t = a_1, o_{t+1} = o_2)|\mathcal{H}_t] = p(A_t = a_1, O_{t+1} = o_2|\mathcal{H}_t).$$

The next value function in the chain uses the previous predictions value $c_1(o_t, a_t, o_{t+1}, H_t) = \mathbb{1}\{A_t = a_0, O_{t+1} = o_1 V_2(\{\mathcal{H}_t A_t, O_{t+1}\})\}$. The resulting value function will predict

$$\begin{aligned}
V_1(\mathcal{H}_t) &= \mathbb{E}[\mathbb{1}(A_t = a_0, O_{t+1} = o_1)\mathbb{E}[\mathbb{1}(A_{t+1} = a_1, O_{t+2} = o_2)|\mathcal{H}_t, A_t, O_{t+1}]|\mathcal{H}_t] \\
&= \sum \mathbb{1}(A_t = a_0, O_{t+1} = o_1)p(A_t, O_{t+1}|\mathcal{H}_t) \sum \mathbb{1}(A_{t+1} = a_1, O_{t+2} = o_2)p(A_{t+1}, O_{t+2}|\mathcal{H}_t, A_t, O_{t+1}) \\
&= \sum\sum \mathbb{1}(A_t = a_0, O_{t+1} = o_1)\mathbb{1}(A_{t+1} = a_1, O_{t+2} = o_2)p(A_t, O_{t+1}|\mathcal{H}_t)p(A_{t+1}, O_{t+2}|\mathcal{H}_t, A_t, O_{t+1}) \\
&= \sum\sum \mathbb{1}(A_t = a_0, O_{t+1} = o_1, A_{t+1} = a_1, O_{t+2} = o_2)p(A_t, O_{t+1}|\mathcal{H}_t)\frac{p(A_t, O_{t+1}, A_{t+1}, O_{t+2}|\mathcal{H}_t)}{p(A_t, O_{t+1}|\mathcal{H}_t)} \\
&= \sum\sum \mathbb{1}(A_t = a_0, O_{t+1} = o_1, A_{t+1} = a_1, O_{t+2} = o_2)p(A_t, O_{t+1}, A_{t+1}, O_{t+2}|\mathcal{H}_t) \\
&= p(A_t = a_0, O_{t+1} = o_1, A_{t+1} = a_1, O_{t+2} = o_2|\mathcal{H}_t)
\end{aligned}$$

□

4.6 Summary

This chapter presented the GVFN predictive representations of state and contrasts this approach to various types of representations of state including recurrent neural networks, predictive state representations, and TD Networks. This chapter concluded by providing some constructions of GVFNs corresponding to state in compass world, k-forecasts, and finally the core-tests of a PSR.

The aim of this chapter was to lay the ground work needed for the next sections, and to discuss the intuitive motivation behind the GVFN approach to state construction. This is the beginning of our exploration into how off-policy prediction can be applied to state construction using recurrent networks, and later chapters unwrap the implications of this approach. The motivation behind the GVFN work is extremely similar to that of TD networks and PSRs, except sufficiency is explicitly considered to be tied only to an agent's goals rather than all possible predictions provided by the agent. While this makes intuitive sense from the perspective of searching and learning the agent state, proving viability becomes intractable from these constraints (Subramanian et al. 2022). The subsequent chapters will focus on the question of usefulness of such an approach through empirical means.

Chapter 5

Learning Algorithms for GVFNs

In this section, I introduce the objective function for GVFNs, that constrains the learned state to be GVF predictions. Each state component of a GVFN is a value function prediction, and so is approximating the fixed point to a Bellman equation with history in Equation (4.3). The extension is not as simple as using a standard Bellman operator, however, because the GVFs are in a network. In fact, the Bellman equations are coupled in two ways: through composition—where one GVF can be the cumulant for another GVF as seen in section 6.3—and through the parametric recurrent state representation. I first discuss the Bellman network operator in Section 5.1, which extends the typical Bellman operator to allow for composition. I then explain how the coupling that arises from the recurrent state representation can be handled using a projected operator, and provide the objective for GVFNs, called the *Mean-Squared Projected Bellman Network Error* (MSPBNE), in Section 5.2. Then I discuss several algorithms to optimize this objective in Section 5.3. Finally, I also describe a simple procedure for using eligibility traces when using semi-gradient temporal-difference methods for GVFNs.

The GVFN objective can be mixed with the standard RNN objective, to provide an RNN where the learned states are both useful for prediction of the target and encouraged—or regularized—to GVF predictions. In this work, GVFNs are only trained with the GVFN objective, without including the loss to a target, to focus the investigation on the utility of the proposed objective and on predictive features.

5.1 The Bellman Network Operator

To understand the Bellman network operator, it is useful to first revisit the Bellman operator for learning a single GVF. The set of histories $\mathcal{H}$ is assumed finite.

Assume a tabular encoding for the values, $\mathbf{V}^{(j)} \in \mathbb{R}^{|\mathcal{H}|}$, for a GVF question $(\pi^{(j)}, c^{(j)}, \gamma^{(j)})$. The Bellman equation in (4.3) can be written as a fixed point equation, with Bellman operator

$$\mathbf{B}^{(j)}\mathbf{V}^{(j)} \stackrel{\mathsf{def}}{=} \mathbf{C}^{(j)} + \mathbf{P}^{(j)}\mathbf{V}^{(j)} \tag{5.1}$$

where $\mathbf{C}^{(j)} \in \mathbb{R}^{|\mathcal{H}|}$ is the vector of expected cumulant values under $\pi^{(j)}$, with entries

$$\mathbf{C}^{(j)}(\mathbf{h}_t) \stackrel{\mathsf{def}}{=} \sum_{a_t \in \mathcal{A}} \pi^{(j)}(a_t|\mathbf{h}_t) \sum_{\mathbf{h}_{t+1} \in \mathcal{H}} p(\mathbf{h}_{t+1}|\mathbf{h}_t, a_t) c^{(j)}(\mathbf{h}_t, a_t, \mathbf{h}_{t+1}). \tag{5.2}$$

and $\mathbf{P}^{(j)} \in \mathbb{R}^{|\mathcal{H}|\times|\mathcal{H}|}$ is the matrix of values satisfying

$$\mathbf{P}^{(j)}(\mathbf{h}_t, \mathbf{h}_{t+1}) = \sum_{a_t \in \mathcal{A}} \pi^{(j)}(a_t|\mathbf{h}_t) p(\mathbf{h}_{t+1}|\mathbf{h}_t, a_t) \gamma^{(j)}(\mathbf{h}_t, a_t, \mathbf{h}_{t+1}). \tag{5.3}$$

If the operator $\mathbf{B}^{(j)}$ is a contraction, then iteratively applying this operator converges to a fixed point. More precisely, if for any $\mathbf{V}_1^{(j)}, \mathbf{V}_2^{(j)} \in \mathbb{R}$, $\|\mathbf{B}^{(j)}\mathbf{V}_1^{(j)} - \mathbf{B}^{(j)}\mathbf{V}_2^{(j)}\| < \|\mathbf{V}_1^{(j)} - \mathbf{V}_2^{(j)}\|$, then iteratively applying $\mathbf{B}^{(j)}$, as $\mathbf{V}_2^{(j)} = \mathbf{B}^{(j)}\mathbf{V}_1^{(j)}, \ldots, \mathbf{V}_{t+1}^{(j)} = \mathbf{B}^{(j)}\mathbf{V}_t^{(j)}, \ldots$, converges to a fixed point. Because temporal difference learning algorithms are based on this fixed-point update, the Bellman operator is central to the analysis of many algorithms for learning value functions, and is used in the definition of objectives for value estimation.

Similarly, a Bellman operator can be defined that accounts for the relationships between GVFs in the network. Assume there are N GVFs, with $\mathbf{V} \in \mathbb{R}^{N|\mathcal{H}|}$ the stacked values for all the GVFs,

$$\mathbf{V} \stackrel{\mathsf{def}}{=} \begin{bmatrix} \mathbf{V}^{(1)} \\ \vdots \\ \mathbf{V}^{(N)} \end{bmatrix}. \tag{5.4}$$

The cumulants may now be functions of the values of other GVFs; therefore cumulants are explicitly written as $\mathbf{C}_{\mathbf{V}}^{(j)}$. The Bellman network operator $\mathbf{B}$ is

$$\mathbf{B}\mathbf{V} \overset{\text{def}}{=} \begin{bmatrix} \mathbf{C}_{\mathbf{V}}^{(1)} + \mathbf{P}^{(1)}\mathbf{V}^{(1)} \\ \vdots \\ \mathbf{C}_{\mathbf{V}}^{(N)} + \mathbf{P}^{(N)}\mathbf{V}^{(N)} \end{bmatrix}. \tag{5.5}$$

The Bellman network operator needs to be treated as a joint operator on all the GVFs because of compositional predictions, where the prediction on the next step of GVF j is the cumulant for GVF i. When iterating the Bellman operator $\mathbf{V}^{(j)}$ is not only involved in its own Bellman equation, but also in the Bellman equation for $\mathbf{V}^{(i)}$. Notice that if there were no compositions, the Bellman network operator would separate into individual Bellman operators, that operate on each $\mathbf{V}^{(j)}$ independently.

To use such a Bellman network operator, it needs to be ensured that iterating under this operator converges to a fixed point. For no composition, this result is straightforward, as it simply follows from previous results showing when the Bellman operator is a contraction. This is stated explicitly below in Corollary 5.1.1.1. Under composition, the effect of the current value function on the cumulant needs to be considered. Consequently, the operator may no longer be a simple linear projection of the values, followed by a sum of expected cumulants.

First, identify a necessary condition: the connections between GVFs must be acyclic. For example, GVF i cannot be a cumulant for GVF j, if j is already a cumulant for i. More generally, the connections between GVFs cannot create a cycle, such as $1 \to 2 \to 3 \to 1$. A counterexample is provided, where the Bellman network operator is not a contraction when there is a cycle, to illustrate that this condition is necessary.

Further restrictions are placed on the cumulant, if it is a function of other GVFs. In particular, it is required that the cumulant is a Lipschitz function of the other value functions. Note that this restriction encompasses the setting for a non-compositional GVF, because the cumulant can be a constant w.r.t. these values. It also encompasses the setting used in the below experiments: that each cumulant is a linear function of the GVF values on the next step.

Assumption 1 (Acyclic Connections)**.** *The directed graph G is acyclic. G consists of N vertices, each corresponding to a GVF, and each directed edge (i, j) indicates that j is used in the cumulant for i.*

Assumption 2 (Lipschitz Compositional Cumulants)**.** *If GVF i has directed edges to $\{j_1, \ldots, j_k\}$, then the cumulant $c^{(i)}_{\mathbf{V}}(\mathbf{h}_{t+1})$ is Lipschitz in $\mathbf{V}^{(j_1)}, \ldots, \mathbf{V}^{(j_k)}$ with Lipschitz constant K_i. That is, for $\mathbf{V}_1, \mathbf{V}_2 \in \mathbb{R}^{N|\mathcal{H}|}$, $\|\mathbf{C}^{(i)}_{\mathbf{V}_1} - \mathbf{C}^{(i)}_{\mathbf{V}_2}\| \leq K_i \sum_{l=1}^{k} \|\mathbf{V}_1^{(j_l)} - \mathbf{V}_2^{(j_l)}\|$.*

Note that this assumption is satisfied if when assuming that for some bounded weights $w_1, \ldots, w_k \in \mathbb{R}$, the cumulant must satisfy $c^{(i)}_{\mathbf{V}}(\mathbf{h}_{t+1}) = \sum_{l=1}^{k} w_l \mathbf{V}^{(j_l)}(\mathbf{h}_{t+1})$ or equivalently, $\mathbf{C}^{(i)}_{\mathbf{V}} = \sum_{l=1}^{k} w_l \mathbf{P}^{(j_l)} \mathbf{V}^{(j_l)}$. This is because $\mathbf{P}^{(j_l)}$ is a non-expansion, and so

$$\|\mathbf{C}^{(i)}_{\mathbf{V}_1} - \mathbf{C}^{(i)}_{\mathbf{V}_2}\| = \left\| \sum_{l=1}^{k} w_l \mathbf{P}^{(j_l)} (\mathbf{V}_1^{(j_l)} - \mathbf{V}_2^{(j_l)}) \right\| \leq \sum_{l=1}^{k} |w_l| \|\mathbf{P}^{(j_l)} (\mathbf{V}_1^{(j_l)} - \mathbf{V}_2^{(j_l)})\|$$
$$\leq (\max_l |w_l|) \sum_{l=1}^{k} \|\mathbf{V}_1^{(j_l)} - \mathbf{V}_2^{(j_l)}\|.$$

The third assumption is standard for showing Bellman operators are contractions, and is easily satisfied if the policy is proper: is guaranteed to visit at least one state where the continuation is less than 1.

Assumption 3 (Discounted Transitions are Contractions)**.** *For all $j \in \{1, \ldots, N\}$, $\beta_j \overset{\text{def}}{=} \|\mathbf{P}^{(j)}\| < 1$, where $\|\cdot\|$ is the spectral norm.*

With these three assumptions, the main result can be proven.

Theorem 5.1.1. *Under Assumptions 1-3, iterating $\mathbf{V}_{t+1} = \mathbf{B}\mathbf{V}_t$ converges to a unique fixed point.*

Proof. First, the proof that the sequence of value estimates converges (Part 1) is detailed, then follows a proof that it converges to a unique fixed point (Part 2 and 3).
Part 1: *{*The sequence $\mathbf{V}_1, \mathbf{V}_2, \ldots$ defined by $\mathbf{V}_{t+1} = \mathbf{B}\mathbf{V}_t$ converges to a limit $\mathbf{V}^* \in \mathbb{R}^{N|\mathcal{H}|}$.*}*

Because G is acyclic, there is a linear topological ordering of the vertices, $i_1, \ldots, i_N$: for each directed edge (i, j), i comes before j in the ordering. Therefore, starting from the last GVF $j = i_N$, remember that the Bellman operator $\mathbf{B}^{(j)}$ is a contraction with rate $\beta_j < 1$,

$$\|\mathbf{B}^{(j)}\mathbf{V}_1^{(j)} - \mathbf{B}^{(j)}\mathbf{V}_0^{(j)}\| = \|\mathbf{P}^{(j)}\mathbf{V}_1^{(j)} - \mathbf{P}^{(j)}\mathbf{V}_0^{(j)}\| \leq \beta_j \|\mathbf{V}_1^{(j)} - \mathbf{V}_0^{(j)}\|.$$

Therefore, iterating $\mathbf{B}$ for t steps results in the error

$$\|\mathbf{V}_{t+1}^{(j)} - \mathbf{V}_t^{(j)}\| \leq \beta_j^t \|\mathbf{V}_1^{(j)} - \mathbf{V}_0^{(j)}\|$$

and as $t \to \infty$, $\mathbf{V}_t^{(j)}$ converges to its fixed point.

Induction is used for the argument, with the above as the base case. Assume for all $j \in \{i_k, \ldots, i_N\}$ there exists a ball of radius $\epsilon(t)$ where $\|\mathbf{V}_{t+1}^{(j)} - \mathbf{V}_t^{(j)}\| \leq \epsilon(t)$ and $\epsilon(t) \to 0$ as $t \to \infty$. Consider the next GVF in the ordering, $i = i_{k-1}$.

Case 1: There are no outgoing edges from i. If i does not use another GVF j in its cumulant, then iterating with $\mathbf{B}$ independently iterates $\mathbf{V}_t^{(i)}$ with $\mathbf{B}^{(i)}$. Therefore, as above, $\mathbf{V}_t^{(i)}$ converges because the Bellman operator is a contraction. In this setting, clearly such an $\epsilon_i(t)$ exists because $\|\mathbf{V}_{t+1}^{(j)} - \mathbf{V}_t^{(j)}\| \to 0$ as $t \to \infty$.

Case 2: The cumulant for GVF i is composed of the values for the set of GVFs $\mathcal{J} \subseteq \{i_k, \ldots, i_N\}$. The basic idea, formalized below, is that GVF i will be guaranteed to converge once the GVFs used to construct the become sufficiently accurate. The update is $\mathbf{V}_{t+1}^{(i)} = \mathbf{C}_{\mathbf{V}_t}^{(i)} + \mathbf{P}^{(i)}\mathbf{V}_t^{(i)}$. The change in $\mathbf{V}_t^{(i)}$ is

$$\begin{aligned}\|\mathbf{V}_{t+1}^{(i)} - \mathbf{V}_t^{(i)}\| &= \|(\mathbf{C}_{\mathbf{V}_t}^{(i)} - \mathbf{C}_{\mathbf{V}_{t-1}}^{(i)}) + \mathbf{P}^{(i)}(\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)})\| \\ &\leq K_i \sum_{j \in \mathcal{J}} \|\mathbf{V}_t^{(j)} - \mathbf{V}_{t-1}^{(j)}\| + \beta_i \|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\| \\ &\leq N K_i \epsilon(t-1) + \beta_i \|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\|.\end{aligned}$$

In the first inequality, the first term is due to Lipschitz continuity of the cumulant and the second term is due to the fact that $\|\mathbf{P}^{(i)}\| = \beta_i$. In the second inequality, $\|\mathbf{V}_t^{(j)} - \mathbf{V}_{t-1}^{(j)}\| \leq \epsilon_j(t)$, under the inductive hypothesis. The second inequality is loose, as the sum only involves $|\mathcal{J}| < N$ terms, but

N is used for simplicity since the results goes through with this constant as well. For sufficiently large t, $\epsilon(t-1)$ can be made arbitrarily small. If $NK_i\epsilon(t-1) < (1-\beta_i)\|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\|$, i.e., $\epsilon(t-1) < \frac{(1-\beta_i)}{NK_i}\|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\|$ then

$$\|\mathbf{V}_{t+1}^{(i)} - \mathbf{V}_t^{(i)}\| \leq \tilde{\beta}_i\|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\| \qquad \text{for some } \tilde{\beta}_i < 1$$

and so the iteration is a contraction on step t. Else, if $\epsilon(t-1) \geq \frac{(1-\beta_i)}{NK_i}\|\mathbf{V}_t^{(i)} - \mathbf{V}_{t-1}^{(i)}\|$, then this implies the difference $\|\mathbf{V}_{t+1}^{(i)} - \mathbf{V}_t^{(i)}\|$ is already within a small ball, with radius $NK_i\epsilon(t-1)/(1-\beta_i)$. As $t \to \infty$, the difference can oscillate between being within this ball, which shrinks to zero because $\epsilon(t)$ shrinks to zero, or being iterated with a contraction that also shrinks the difference. In either case, there exists an $\epsilon_i(t)$ such that $\|\mathbf{V}_{t+1}^{(i)} - \mathbf{V}_t^{(i)}\| \leq \epsilon_i(t)$, where $\epsilon_i(t) \to 0$ as $t \to \infty$.

By induction, there exists such a ϵ_i for all GVFs in the network. Therefore, the sequence $\mathbf{V}_t^{(i)}$ converges.

Part 2: $\mathbf{V}^*$ *is a fixed point of* $\mathbf{B}$.

Because the Bellman network operator is continuous, the limit can be taken inside the operator

$$\mathbf{V}^* = \lim_{t\to\infty} \mathbf{V}_t = \lim_{t\to\infty} \mathbf{B}\mathbf{V}_{t-1} = \mathbf{B}\left(\lim_{t\to\infty} \mathbf{V}_{t-1}\right) = \mathbf{B}\mathbf{V}^*$$

Part 3: $\mathbf{V}^*$ *is the only fixed point of* $\mathbf{B}$.

Consider an alternative solution $\mathbf{V}$. Because of the uniqueness of fixed points under Bellman operators, all those GVFs that have non-compositional cumulants have unique fixed points and so those components in $\mathbf{V}$ must be the same as $\mathbf{V}^*$. All the GVFs next in the ordering that use those GVFs as cumulants must then also converge to a unique value, because their Bellman operators with fixed GVFs as cumulants have a unique fixed point. This argument continues for the remaining GVFs in the ordering. □

Corollary 5.1.1.1. *Under Assumption 3 with non-compositional cumulants (no edges in G), iterating* $\mathbf{V}_{t+1} = \mathbf{B}\mathbf{V}_t$ *converges to a unique fixed point.*

Proposition 5.1.2 (Necessity of Acyclic Composition). *There exists transition function* $\mathrm{P} : \boldsymbol{\Psi} \times \mathcal{A} \times \boldsymbol{\Psi} \to [0,1]$ *and policy* $\pi : \boldsymbol{\Psi} \times \mathcal{A} \to [0,1]$ *such that, for two GVFs in a cycle, iteration with the Bellman network operator diverges.*

Proof. Assume there are two states, with the policy defined such that the Markov chain has the following dynamics

$$\mathbf{P}^{\pi} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}. \tag{5.6}$$

Assume further that $\gamma = 0.95$. The resulting Bellman iteration is

$$\begin{aligned}
\begin{bmatrix}\mathbf{V}^{(1)} \\ \mathbf{V}^{(2)}\end{bmatrix} &= \mathbf{P}^{\pi}\begin{bmatrix}\mathbf{V}^{(2)} \\ \mathbf{V}^{(1)}\end{bmatrix} + \gamma\mathbf{P}^{\pi}\begin{bmatrix}\mathbf{V}^{(1)} \\ \mathbf{V}^{(2)}\end{bmatrix} \\
&= \mathbf{P}^{\pi}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix}\mathbf{V}^{(1)} \\ \mathbf{V}^{(2)}\end{bmatrix} + \mathbf{P}^{\pi}\begin{bmatrix} \gamma & 0 \\ 0 & \gamma \end{bmatrix}\begin{bmatrix}\mathbf{V}^{(1)} \\ \mathbf{V}^{(2)}\end{bmatrix} \\
&= \mathbf{P}^{\pi}\begin{bmatrix} \gamma & 1 \\ 1 & \gamma \end{bmatrix}\begin{bmatrix}\mathbf{V}^{(1)} \\ \mathbf{V}^{(2)}\end{bmatrix}
\end{aligned}$$

Since the matrix $\mathbf{P}^{\pi}\begin{bmatrix} \gamma & 1 \\ 1 & \gamma \end{bmatrix}$ is an expansion, for many initial $\begin{bmatrix}\mathbf{V}^{(1)} \\ \mathbf{V}^{(2)}\end{bmatrix}$ this iteration goes to infinity, such as initial $\mathbf{V}^{(1)} = \mathbf{V}^{(2)} = \begin{bmatrix}1 \\ 1\end{bmatrix}$. □

5.2 The Objective Function for GVFNs

With a valid Bellman network operator, the objective function for GVFNs can be defined. The above fixed point equation assumes a tabular setting, where the values can be estimated directly for each history. GVFNs, however, have a restricted functional form, where the value estimates must be a parametrized function of the current observation and value predictions from the last time-step. Under such a functional form, it is unlikely that the fixed point can be exactly solved for. Rather, a projected fixed point will be used, which projects into the space of representable value functions.

Define the space of functions as

$$\begin{aligned}
\mathcal{F} = \Big\{\mathbf{V}_{\mathbf{W}} = [\mathbf{V}_{\mathbf{W}}^{(1)}, \ldots, \mathbf{V}_{\mathbf{W}}^{(N)}] \in \mathbb{R}^{N|\mathcal{H}|} \quad | \quad \text{where } \mathbf{W} \in \mathbf{\Omega} \quad \text{and} \qquad (5.7) \\
V_{\mathbf{W}}(\mathbf{h}_{t+1}) = f_{\mathbf{W}}([V_{\mathbf{W}}^{(1)}(\mathbf{h}_t), \ldots, V_{\mathbf{W}}^{(N)}(\mathbf{h}_t)], \mathbf{x}_{t+1}) \qquad (5.8) \\
\text{when } p(\mathbf{h}_{t+1}|\mathbf{h}_t, \mathbf{x}_{t+1}) > 0\Big\}
\end{aligned}$$

where $V_{\mathbf{W}}$ is used to mean the value function parameterized by weights (i.e. $V(;\mathbf{W})$) to minimize notation. Recall that $\mathbf{x}_{t+1} = [a_t, \mathbf{o}_{t+1}]$. $p(\mathbf{h}_{t+1}|\mathbf{h}_t, \mathbf{x}_{t+1}) > 0$ only when $\mathbf{h}_{t+1} \equiv \mathbf{h}_t a_t \mathbf{o}_{t+1}$, and so expect this to only be true for one outcome $\mathbf{h}_{t+1}$. And write that $\mathbf{h}_{t+1}$ is equivalent, rather than equal, to the current history appended with action a_t and observation $\mathbf{o}_{t+1}$, because $\mathbf{h}_{t+1}$ might be shorter (more minimal): earlier actions and observations might not be needed. Define the projection operator

$$\Pi_{\mathcal{F}}(\mathbf{V}) \overset{\text{def}}{=} \min_{\hat{\mathbf{V}}\in\mathcal{F}} \|\mathbf{V} - \hat{\mathbf{V}}\|_{\mathbf{d}}^2 \qquad \text{where } \|\mathbf{V} - \hat{\mathbf{V}}\|_{\mathbf{d}}^2 \overset{\text{def}}{=} \sum_{\mathbf{h}\in\mathcal{H}} \mathbf{d}(\mathbf{h})(V(\mathbf{h}) - \hat{V}(\mathbf{h}))^2 \tag{5.9}$$

where $\mathbf{d} : \mathcal{H} \to [0, 1]$ is the sampling distribution over histories. Typically, data is assumed to be generated by following a behavior policy $\mu : \mathcal{H} \to [0, 1]$, and that $\mathbf{d}$ is the stationary distribution for this policy. The value functions for policies π_i are typically learned off-policy, since in general π_i will not equal μ. The behavior policy μ used to gather the data is different, or off of, the policy—or policies—that are being evaluated.

To obtain the projected fixed point solution, a natural goal is to minimize the following projected objective,

$$\min_{\mathbf{W}\in\boldsymbol{\Omega}} \|\Pi_{\mathcal{F}}\mathbf{B}\mathbf{V}_{\mathbf{W}} - \mathbf{V}_{\mathbf{W}}\|_{\mathbf{d}}^2 \tag{5.10}$$

Unfortunately, this objective can be hard to compute, because the projection operator $\Pi_{\mathcal{F}}$ onto the nonlinear manifold can be intractable. Instead, the same approach as Maei, Szepesvári, Bhatnagar, and Sutton (2010) is taken, when defining the nonlinear MSPBE for learning value functions with neural networks and other nonlinear function approximators. The idea is to approximate the projection onto the nonlinear manifold by assuming it is locally linear. Then, a linear projection operator can be used, defined locally at the current set of parameters $\mathbf{W} \in \boldsymbol{\Omega}$, spanned by the basis $\boldsymbol{\phi}_{j,\mathbf{W}}(\mathbf{h}) \overset{\text{def}}{=} \nabla_\theta \mathbf{V}_{\mathbf{W}}^{(j)}(\mathbf{h})$ for all $\mathbf{h} \in \mathcal{H}$ and GVFs j. Let $\boldsymbol{\Phi}_{j,\mathbf{W}}$ correspond to the matrix of stacked $\boldsymbol{\phi}_{j,\mathbf{W}}(\mathbf{h})^\top$ for all $\mathbf{h} \in \mathcal{H}$, having $|\mathcal{H}|$ rows. Define

$$\boldsymbol{\Phi}_{\mathbf{W}} \overset{\text{def}}{=} \begin{bmatrix} \boldsymbol{\Phi}_{1,\mathbf{W}} \\ \vdots \\ \boldsymbol{\Phi}_{N,\mathbf{W}} \end{bmatrix} \qquad \mathbf{D} \overset{\text{def}}{=} \operatorname{diag} \begin{bmatrix} \mathbf{d} \\ \vdots \\ \mathbf{d} \end{bmatrix} \qquad \Pi_{\mathbf{W}} \overset{\text{def}}{=} \boldsymbol{\Phi}_{\mathbf{W}}(\boldsymbol{\Phi}_{\mathbf{W}}^\top \mathbf{D}\boldsymbol{\Phi}_{\mathbf{W}})^{-1}\boldsymbol{\Phi}_{\mathbf{W}}^\top \mathbf{D}.$$

Using this locally linear approximation to the objective potentially expands the set of stationary points. The fixed points under the original projection are still fixed points under this locally linear approximation. But, there could be points that are fixed points under this locally linear approximation, that would not be under the original.

The final objective using this projection is called the MSPBNE[1], defined as

$$\text{MSPBNE}(\mathbf{W}) \overset{\text{def}}{=} \|\Pi_{\mathbf{W}}\mathbf{B}\mathbf{V}_{\mathbf{W}} - \mathbf{V}_{\mathbf{W}}\|^2_{\mathbf{d}} \tag{5.11}$$

In the following lemma it is shown, with proof, that in can be rewritten in a way that makes it more amenable to compute and sample gradients.[2] This reformulation is used to develop algorithms to minimize this objective in the next section.

Lemma 5.2.1. *The MSPBNE defined in Equation* (5.11) *can be rewritten as*

$$\mathit{MSPBNE}(\mathbf{W}) = \boldsymbol{\delta}(\mathbf{W})^\top \Lambda(\mathbf{W})^{-1}\boldsymbol{\delta}(\mathbf{W}) \tag{5.12}$$

where

$$\Lambda(\mathbf{W}) \overset{\text{def}}{=} \mathbb{E}_d\left[\sum_{j=1}^{N} \boldsymbol{\phi}_{j,\mathbf{W}}(H)\boldsymbol{\phi}_{j,\mathbf{W}}(H)^\top\right] = \sum_{\mathbf{h}\in\mathcal{H}} d(\mathbf{h}) \sum_{j=1}^{N} \boldsymbol{\phi}_{j,\mathbf{W}}(\mathbf{h})\boldsymbol{\phi}_{j,\mathbf{W}}(\mathbf{h})^\top \tag{5.13}$$

$$\boldsymbol{\delta}(\mathbf{W}) \overset{\text{def}}{=} \sum_{j=1}^{N} \mathbb{E}_{d,\pi_j}\left[\boldsymbol{\delta}_j(H,A,H')\boldsymbol{\phi}_{j,\mathbf{W}}(H)\right]$$

$$\boldsymbol{\delta}_j(H,A,H') \overset{\text{def}}{=} c^{(j)}(H,A,H') + \gamma^{(j)}(H,A,H')V^{(j)}_{\mathbf{W}}(H') - V^{(j)}_{\mathbf{W}}(H).$$

Proof. Starting with equation (5.11) and for $\Delta_{\mathbf{W}} \overset{\text{def}}{=} \mathbf{B}\mathbf{V}_{\mathbf{W}} - \mathbf{V}_{\mathbf{W}}$,

$$\begin{aligned}\text{MSPBNE}(\mathbf{W}) &= \|\Pi_{\mathbf{W}}\mathbf{B}\mathbf{V}_{\mathbf{W}} - \mathbf{V}_{\mathbf{W}}\|^2_{\mathbf{d}}\\ &= \|\Pi_{\mathbf{W}}\left[\mathbf{B}\mathbf{V}_{\mathbf{W}} - \mathbf{V}_{\mathbf{W}}\right]\|^2_{\mathbf{d}}\\ &= \|\Pi_{\mathbf{W}}\Delta_{\mathbf{W}}\|^2_{\mathbf{d}}\end{aligned}$$

[1]A variant of the MSPBNE has been introduced for TD networks (Silver 2013); the above generalizes that MSPBNE to GVF Networks. Because it is a strict generalization, thus the same name is used.

[2]Since developing the MSPBNE, an alternative approach to defining a nonlinear MSPBE has been developed using a conjugate form for the Bellman error (see Dai et al. (2017) and in-preparation work that makes the connection the MSPBE more explicit (Patterson et al. 2022)). The extension here should be relatively straightforward, as the objective is formulated using histories.

The projection operator can be wrapped around the full TD error $\Delta_{\mathbf{W}}$, because it has no affect on $\mathbf{V}_{\mathbf{W}}$ which is already in the space. Then plugging in the definition of $\Pi_{\mathbf{W}}$

$$\begin{aligned}
\Pi_{\mathbf{W}}^\top \mathbf{D} \Pi_{\mathbf{W}} &= \mathbf{D}^\top \mathbf{\Phi}_{\mathbf{W}} (\mathbf{\Phi}_{\mathbf{W}}^\top \mathbf{D} \mathbf{\Phi}_{\mathbf{W}})^{-1} \mathbf{\Phi}^\top \mathbf{D} \\
\|\Pi_{\mathbf{W}} \Delta_{\mathbf{W}}\|_{\mathbf{d}}^2 &= \Delta_{\mathbf{W}}^\top \Pi_{\mathbf{W}}^\top \mathbf{D} \Pi_{\mathbf{W}} \Delta_{\mathbf{W}} \\
&= \Delta_{\mathbf{W}}^\top \mathbf{D}^\top \mathbf{\Phi}_{\mathbf{W}} (\mathbf{\Phi}_{\mathbf{W}}^\top \mathbf{D} \mathbf{\Phi}_{\mathbf{W}})^{-1} \mathbf{\Phi}^\top \mathbf{D} \Delta_{\mathbf{W}} \qquad (5.14)
\end{aligned}$$

As in prior gradient TD work the matrix operations are converted to expectation forms.

$$\begin{aligned}
\mathbf{\Phi}_{\mathbf{W}}^\top \mathbf{D} \mathbf{\Phi}_{\mathbf{W}} &= \sum_{j=1}^{n} \sum_{\mathbf{h} \in \mathcal{H}} \mathbf{d}(\mathbf{h}) \boldsymbol{\phi}_{j,\mathbf{W}}(\mathbf{h}) \phi_{j,\mathbf{W}}(\mathbf{h})^\top \\
&= \mathbb{E}_d \left[\sum_{j=1}^{n} \boldsymbol{\phi}_{j,\mathbf{W}}(H) \boldsymbol{\phi}_{j,\mathbf{W}}(H)^\top \right] \\
&= W(\mathbf{W}) \\
\mathbf{\Phi}_{\mathbf{W}}^\top \mathbf{D} \Delta_{\mathbf{W}} &= \sum_{j=1}^{n} \sum_{\mathbf{h} \in \mathcal{H}} \mathbf{d}(\mathbf{h}) \boldsymbol{\phi}_{j,\theta}(\mathbf{h}) \sum_{a \in \mathcal{A}} \pi_j(a|\mathbf{h}) \mathbb{E}[\delta_j(\mathbf{h}, a, H')] \\
&= \sum_{j=1}^{n} \mathbb{E}_{d,\pi_j} \left[\delta_j(H, A, H') \boldsymbol{\phi}_{j,\mathbf{W}}(H) \right] \\
&= \boldsymbol{\delta}(\mathbf{W})
\end{aligned}$$

Then substituting into equation (5.14), getting the result $\text{MSPBNE}(\mathbf{W}) = \boldsymbol{\delta}(\mathbf{W})^\top \Lambda(\mathbf{W})^{-1} \boldsymbol{\delta}(\mathbf{W})$. □

Samples are not received according to π_j; instead, they are sampled according to the behavior μ. Throughout this work, it has been assumed μ has a coverage property. This means that the behavior policy μ satisfies $\mu(a|\mathbf{h}) > 0$ if any $\pi_j(a|\mathbf{h}) > 0$ for policies $\pi_1, \ldots, \pi_N$.

Corollary 5.2.1.1. *For importance sampling ratios* $\rho_j(a|\mathbf{h}) \overset{\text{def}}{=} \frac{\pi_j(a|\mathbf{h})}{\mu(a|\mathbf{h})}$ *and*

$$\begin{aligned}
\boldsymbol{\delta}_\mu(\mathbf{W}) &\overset{\text{def}}{=} \mathbb{E}_{d,\mu} \Bigg[\sum_{j=1}^{N} \rho_j(H, A) \boldsymbol{\delta}_j(H, A, H') \boldsymbol{\phi}_{j,\mathbf{W}}(H) \Bigg] \\
&= \sum_{\mathbf{h} \in \mathcal{H}} d(\mathbf{h}) \sum_{a \in \mathcal{A}} \mu(a|\mathbf{h}) \sum_{j=1}^{N} \rho_j(a|\mathbf{h}) \mathbb{E} \Bigg[\boldsymbol{\delta}_j(H, A, H') \boldsymbol{\phi}_{j,\mathbf{W}}(\mathbf{h}) | H = \mathbf{h}, A = a \Bigg]
\end{aligned}$$

then it can be shown that $\boldsymbol{\delta}_\mu(\mathbf{W}) = \boldsymbol{\delta}(\mathbf{W})$ *and so*

$$\mathit{MSPBNE}(\mathbf{W}) = \boldsymbol{\delta}_\mu(\mathbf{W})^\top \Lambda(\mathbf{W})^{-1} \boldsymbol{\delta}_\mu(\mathbf{W})$$

Proof. The key is simply to show that $\boldsymbol{\delta}_\mu(\mathbf{W}) = \boldsymbol{\delta}(\mathbf{W})$, because $W(\mathbf{W})$ depends only on d, not on the policies π or μ. This is straightforward with the typical cancellation in importance sampling ratios

$$\begin{aligned}
\boldsymbol{\delta}_\mu(\mathbf{W}) &= \sum_{\mathbf{h}\in\mathcal{H}} d(\mathbf{h}) \sum_{a\in\mathcal{A}} \mu(a|\mathbf{h}) \sum_{j=1}^{N} \rho_j(a|\mathbf{h}) \mathbb{E}\Big[\boldsymbol{\delta}_j(\mathbf{h}, a, H')\boldsymbol{\phi}_{j,\mathbf{W}}(h) | H = h, A = a\Big] \\
&= \sum_{\mathbf{h}\in\mathcal{H}} d(\mathbf{h}) \sum_{j=1}^{N} \sum_{a\in\mathcal{A}} \mu(a|\mathbf{h}) \rho_j(a|\mathbf{h}) \mathbb{E}\Big[\boldsymbol{\delta}_j(\mathbf{h}, a, H')\boldsymbol{\phi}_{j,\mathbf{W}}(h) | H = h, A = a\Big] \\
&= \sum_{\mathbf{h}\in\mathcal{H}} d(\mathbf{h}) \sum_{j=1}^{N} \sum_{a\in\mathcal{A}} \pi_j(a|\mathbf{h}) \mathbb{E}\Big[\boldsymbol{\delta}_j(\mathbf{h}, a, H')\boldsymbol{\phi}_{j,\mathbf{W}}(h) | H = h, A = a\Big] \\
&= \boldsymbol{\delta}(\mathbf{W}).
\end{aligned}$$

□

From here on, therefore, it is assumed that $\boldsymbol{\delta}(\mathbf{W})$ is defined more generally as the above $\boldsymbol{\delta}_\mu(\mathbf{W})$, since they result in the same objective but this more general expression more obviously highlights off-policy sampling.

From this reformulation, one can see that the MSPBNE objective is a weighted quadratic objective, with weighting matrix $W(\mathbf{W})$ on vector $\boldsymbol{\delta}(\mathbf{W})$. The objective is zero—and so minimal—when $\boldsymbol{\delta}(\mathbf{W}) = \mathbf{0}$. This is similar to the temporal difference (TD) learning fixed point criteria. In fact, TD implicitly optimizes the linear MSPBE, which corresponds to the above objective with $N = 1$ and fixed features that do not depend on the parameters. Once we have a projected Bellman error objective, we can take advantage of the many advances in formulating TD algorithms to optimized MSPBE objectives. Therefore, though this objective looks quite complex, there is substantial literature to facilitate minimizing the MSPBNE.

5.3 Algorithms for the MSPBNE

The algorithms to optimize the MSPBNE are a relatively straightforward combination of standard algorithms for RNNs and the TD algorithms designed to optimize the MSPBE. To provide some intuition on these algorithms, and how to obtain this combination of TD and RNN algorithms, I begin with a simpler setting: extending TD to a recurrent setting, with one GVF. From there, two algorithms for the MSPBNE are introduced: Recurrent TD and Recurrent GTD.

Consider first the on-policy TD update, without recurrence, assuming the true state $\mathbf{s}_t$ at time t is given:

$$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t + \alpha_t \delta_t \nabla_{\mathbf{W}} V(\mathbf{s}_t; \mathbf{W}_t)$$
$$\text{where } \delta_t \overset{\text{def}}{=} C_{t+1} + \gamma_{t+1} V(\mathbf{s}_{t+1}; \mathbf{W}_t) - V(\mathbf{s}_t; \mathbf{W}_t).$$

With recurrence, where the state is estimated and so is a function of $\mathbf{W}_t$, the only difference to this update is in the computation of $\nabla_{\mathbf{W}} V(\mathbf{s}_t; \mathbf{W}_t)$, where $\mathbf{s}_t$ should instead be thought of $\mathbf{s}_t(\mathbf{W}_t)$. This gradient now requires the chain rule, to account for the impact of $\mathbf{W}_t$ on the last state, and the state before then, and so on:

$$\frac{\partial V(\mathbf{s}_t; \mathbf{W}_t)}{\partial \mathbf{W}_i} = \frac{\partial V(\mathbf{s}_t; \mathbf{W}_t)}{\partial \mathbf{s}_t}^\top \frac{\partial \mathbf{s}_t}{\partial \mathbf{W}_i}$$

where $\mathbf{s}_t = f_{\mathbf{W}}(\mathbf{s}_{t-1}, \mathbf{x}_t)$. Computing this gradient back-in-time, $\nabla_{\mathbf{W}} \mathbf{s}_t$—which is also called the *sensitivity*—is precisely the aim of most RNN algorithms, including truncated BPTT and RTRL. Any algorithm that computes sensitivities can be used to obtain a TD update with recurrent connections to estimate the state.

For GVFNs, there are two differences: one needs to account for off-policy sampling and the fact that state is itself composed of these value estimates, rather than being learned to estimate values. Value estimation within GVFNs requires off-policy updates, because the target policies π_j are not typically equal to the behavior policy μ. Therefore, importance sampling ratios must be included in the update

$$\rho_{t,j} \overset{\text{def}}{=} \frac{\pi_j(A_t|\mathbf{h}_t)}{\mu(A_t|\mathbf{h}_t)} \qquad \text{for all } j \in \{1, 2, \ldots, N\}.$$

This ratio multiplies the TD update, to adjust the expectation of the update to be as if action A_t had been taken under π_j rather than the behavior μ. For the second difference, the Recurrent TD update is actually even simpler because the value function itself is the state. For the j-th value function—which is the j-th state variable—$\nabla_{\mathbf{W}} V^{(j)}(;\mathbf{W})$ at time t is $\nabla_{\mathbf{W}} \mathbf{s}_{t,j}$. Notice that this gradient actually corresponds to using the above chain rule update, by using $V^{(j)}(\mathbf{s}_t) = \mathbf{s}_{t,j}$ as a selector function into the state variable.

The **Recurrent TD** update for GVFNs is

$$\begin{aligned}
\mathbf{s}_t &\leftarrow f(\mathbf{s}_{t-1}, \mathbf{x}_t; \mathbf{W}_t) && \triangleright \text{ where } \mathbf{x}_t \overset{\text{def}}{=} [a_{t-1}, \mathbf{o}_t] \\
\mathbf{s}_{t+1} &\leftarrow f(\mathbf{s}_t, \mathbf{x}_{t+1}; \mathbf{W}_t) && \triangleright \text{ where } \mathbf{x}_{t+1} \overset{\text{def}}{=} [a_t, \mathbf{o}_{t+1}] \\
\boldsymbol{\phi}_{t,j} &\leftarrow \nabla_{\mathbf{W}} \mathbf{s}_{t,j} && \triangleright \text{ Compute sensitivities using p-BPTT} \\
\delta_{t,j} &\leftarrow C^{(j)}_{t+1} + \gamma^{(j)}_{t+1} \mathbf{s}_{t+1,j} - \mathbf{s}_{t,j} && \\
\rho_{t,j} &\leftarrow \frac{\pi^{(j)}(a_t|\mathbf{o}_t)}{\mu(a_t|\mathbf{o}_t)} && \triangleright \text{ Policies can be functions of histories.} \\
\mathbf{W}_{t+1} &\leftarrow \mathbf{W}_t + \alpha_t \left[\sum_{j=1}^{N} \rho_{t,j} \boldsymbol{\delta}_{t,j} \boldsymbol{\phi}_{t,j} \right] &&
\end{aligned}$$

The TD update, however, is only an approximate semi-gradient update, even in the fully observable setting. To obtain exact gradient formulas, I turn to Gradient TD (GTD) algorithms. In particular, the nonlinear GTD strategy developed by Maei, Szepesvári, Bhatnagar, Silver, et al. 2009 to the MSPBNE. As above, any algorithm to compute the sensitivities can be used in the Recurrent GTD algorithm. But, the algorithm becomes more complex, simply because nonlinear GTD is more complex than TD even without recurrence.

The following theorem is used to facilitate estimating the gradient. The main idea is to introduce an auxiliary weight vector, $\mathbf{u}$, to provide a quasi-stationary estimate of part of the objective. This proof and explicit derivation for the resulting Recurrent TD algorithm is given in the Section 5.4. As a warm up, the result for non-compositional GVFs is derived: no GVFs predict the outcomes of other GVFs. This makes the algorithm easier to follow. The more general proof and derivation is contained in Section 5.4.

Theorem 5.3.1. *Assume that* $V(\mathbf{h};\mathbf{W})$ *is twice continuously differentiable as a function of* $\mathbf{W}$ *for all histories* $\mathbf{h} \in \mathcal{H}$ *where* $\mathbf{d}(\mathbf{h}) > 0$ *and that* $W(\cdot)$, *defined in Equation* (5.13), *is non-singular in a small neighbourhood of* $\mathbf{W}$. *Assume further that there are no compositional GVFs in the GVFN: no GVFs has a cumulant that corresponds to another GVFs prediction. Then for* $\mathbf{u}(\mathbf{W})$ *and* $\boldsymbol{\delta}(\mathbf{W})$ *defined in Lemma 5.2.1,*

$$\mathbf{u}(\mathbf{W}) \overset{\text{def}}{=} \Lambda(\mathbf{W})^{-1}\boldsymbol{\delta}(\mathbf{W}) \tag{5.15}$$

$$\hat{\delta}_{j,\theta}(H) \overset{\text{def}}{=} \boldsymbol{\phi}_{j,\mathbf{W}}(H)^\top \mathbf{u}(\mathbf{W})$$

$$\boldsymbol{\psi}(\mathbf{W}) \overset{\text{def}}{=} \mathbb{E}_{d,\mu}\left[\sum_{j=1}^{N} \rho_j(H,A)\Big(\delta_j(H,A,H') - \hat{\delta}_{j,\theta}(H)\Big)\nabla^2 V^{(j)}_{\mathbf{W}}(H)\mathbf{u}(\mathbf{W})\right] \tag{5.16}$$

getting the gradient

$$-\tfrac{1}{2}\nabla \mathit{MSPBNE}(\mathbf{W}) = \boldsymbol{\delta}(\mathbf{W}) - \mathbb{E}_{d,\mu}\Big[\rho_j(H,A)\gamma^{(j)}(H,A,H')\hat{\delta}_{j,\theta}(H)\boldsymbol{\phi}_{j,\mathbf{W}}(H')\Big] - \boldsymbol{\psi}(\mathbf{W})$$

Now two additional terms are needed to estimate beyond the standard sensitivities in a typical RNN gradient. First, the additional weight vector $\mathbf{u}$ is estimated via Equation (5.15). This can be done using standard techniques in reinforcement learning. Second, a Hessian-vector product must be estimated. The Hessian-vector product is given in Equation (5.16). Fortunately, this can be computed using R-operators, without explicitly computing the Hessian-vector product, using only computation linear in the length of the vector.

The **Recurrent GTD** update, for this simpler setting without composition, is[3]

$$
\begin{aligned}
\mathbf{s}_t &\leftarrow f(\mathbf{s}_{t-1}, \mathbf{x}_t; \mathbf{W}_t) \\
\mathbf{s}_{t+1} &\leftarrow f(\mathbf{s}_t, \mathbf{x}_{t+1}; \mathbf{W}_t) \\
\boldsymbol{\phi}_{t,j} &\leftarrow \nabla_{\mathbf{W}} \mathbf{s}_{t,j} && \triangleright \text{ Compute sensitivities using truncated BPTT} \\
\boldsymbol{\phi}'_{t,j} &\leftarrow \nabla_{\mathbf{W}} \mathbf{s}_{t+1,j} \\
\rho_{t,j} &\leftarrow \frac{\pi^{(j)}(a_t|\mathbf{o}_t)}{\mu(a_t|\mathbf{o}_t)} \\
\mathbf{v}_t &\leftarrow \nabla^2 \mathbf{s}_t \mathbf{u}_t && \triangleright \text{ Computed using R-operators, see Appendix 5.5} \\
\hat{\delta}_{t,j} &\leftarrow \boldsymbol{\phi}_{t,j}^\top \mathbf{u}_t \\
\boldsymbol{\psi}_t &\leftarrow \sum_{j=1}^{N} (\rho_{t,j}\delta_{t,j} - \hat{\delta}_{t,j}) \mathbf{v}_t \\
\mathbf{W}_{t+1} &\leftarrow \mathbf{W}_t + \alpha_t \left[\sum_{j=1}^{N} \rho_{t,j} \boldsymbol{\delta}_{t,j} \boldsymbol{\phi}_{t,j} - \rho_{t,j} \gamma_{j,t+1} \hat{\delta}_{t,j} \boldsymbol{\phi}'_{t,j} \right] - \alpha_t \boldsymbol{\psi}_t \\
\mathbf{u}_{t+1} &\leftarrow \mathbf{u}_t + \beta_t \left[\sum_{j=1}^{N} \rho_{t,j} (\boldsymbol{\delta}_{t,j} - \hat{\delta}_{t,j}) \boldsymbol{\phi}_{t,j} \right]
\end{aligned}
$$

The derivation for this algorithm is similar to the derivation for Gradient TD Networks (Silver 2013), though for this more general setting with GVF Networks.

As alluded to, there are a variety of possible strategies to optimize the MSPBNE for GVFNs. This variety arises from different strategies to optimize RNNs, back-in-time, as well as from the variety of strategies to optimize the MSPBE for value estimation. For example, sensitivities can be computed using truncated BPTT or RTRL and its many approximations. Similarly, for the MSPBE, there are a variety of different strategies to approximate gradients, because the gradient is not straightforward to sample. These including a variety of gradient TD methods—such as GTD and GTD2—saddlepoint methods and semi-gradient TD (see Ghiassian et al. (2018) for a more exhaustive list).

[3]As mentioned above, one could have considered an alternative MSPBNE, using an recently published nonlinear MSPBE objective (Patterson et al. 2022). The resulting Recurrent GTD algorithm would look very similar, except the Hessian-vector product could be omitted: $\boldsymbol{\psi}_t$ is simply dropped in the update to θ.

5.4 Deriving the Full Recurrent GTD Update

Now that the objective is written in its expectation form, the gradients can be take with respect to the weight parameter. The main body stated the result for a simplified setting (Theorem 5.3.1), to make it simpler to understand the result. The more general result, for compositional GVFs, is presented in this section.

Theorem 5.4.1. *Assume that $V_{\mathbf{W}}(\mathbf{h})$ is twice continuously differentiable as a function of $\mathbf{W}$ for all histories $\mathbf{h} \in \mathcal{H}$ where $\mathbf{d}(\mathbf{h}) > 0$ and that $W(\cdot)$, defined in Equation* (5.13)*, is non-singular in a small neighbourhood of $\mathbf{W}$. Then for*

$$\boldsymbol{\delta}(\mathbf{W}) \overset{\text{def}}{=} \mathbb{E}_{d,\mu}\left[\sum_{j=1}^{N} \rho_j(H,A)\boldsymbol{\delta}_j(H,A,H')\boldsymbol{\phi}_{j,\mathbf{W}}(H)\right]$$
$$\mathbf{u}(\mathbf{W}) = W(\mathbf{W})^{-1}\boldsymbol{\delta}(\mathbf{W})$$
$$\boldsymbol{\psi}(\mathbf{W}) = \mathbb{E}_{d,\mu}\left[\sum_{j=1}^{N}\Big(\rho_j(H,A)\delta_j(H,A,H') - \boldsymbol{\phi}_{j,\mathbf{W}}(H)^\top \mathbf{u}(\mathbf{W})\Big)\nabla^2 V^{(j)}_{\mathbf{W}}(H)\mathbf{u}(\mathbf{W})\right]$$

getting the gradient

$$-\frac{1}{2}\nabla MSPBNE(\mathbf{W}) = -\mathbb{E}_{d,\mu}\left[\sum_{j=1}^{N}\rho_j(H,A)\nabla_{\mathbf{W}}\delta_j(H,A,H')\boldsymbol{\phi}_{j,\theta}(H)^\top\right]\mathbf{u}(\mathbf{W}) - \boldsymbol{\psi}(\mathbf{W}) \tag{5.17}$$

Proof. For simplicity in notation below, the explicit dependence on the random variable H are dropped in the expectations.

$$\boldsymbol{\phi}_{j,\mathbf{W}}(H) \to \boldsymbol{\phi}_{j,\mathbf{W}}, \qquad \boldsymbol{\phi}_{j,\mathbf{W}}(H') \to \boldsymbol{\phi}'_{j,\mathbf{W}}$$
$$\boldsymbol{\delta}_j(H,A,H') \to \boldsymbol{\delta}_j, \qquad \rho_j(H,A) \to \rho_j$$

Further, ∂_i indicates the partial derivative with respect to $\mathbf{W}_i$. Also, assume all expectations are with respect to d, and μ. J is used to denote the MSPBNE, which from Lemma 5.2.1 and Corollary 5.2.1.1, can be written $J(\mathbf{W}) = \boldsymbol{\delta}(\mathbf{W})^\top W(\mathbf{W})^{-1}\boldsymbol{\delta}(\mathbf{W})$. When applying the product rule Recall that $\mathbf{u}(\mathbf{W}) =$

$W(\mathbf{W})^{-1}\boldsymbol{\delta}(\mathbf{W})$, and that $W(\mathbf{W})$ is symmetric, giving

$$\begin{aligned}
\partial_i J(\mathbf{W}) &= 2(\partial_i \boldsymbol{\delta}(\mathbf{W}))^\top \mathbf{u}(\mathbf{W}) + \boldsymbol{\delta}(\theta)^\top \partial_i W(\mathbf{W})^{-1} \boldsymbol{\delta}(\theta) \\
\partial_i \boldsymbol{\delta}(\mathbf{W}) &= \mathbb{E}\Big[\sum_{j=1}^{N} \rho_j \partial_i \boldsymbol{\phi}_{j,\mathbf{W}} \boldsymbol{\delta}_j + \boldsymbol{\phi}_{j,\mathbf{W}} \partial_i \boldsymbol{\delta}_j\Big] \\
\partial_i W(\mathbf{W})^{-1} &= -W(\mathbf{W})^{-1} \partial_i W(\mathbf{W}) W(\mathbf{W})^{-1} \\
&= -2W(\mathbf{W})^{-1} \mathbb{E}\Big[\sum_{j=1}^{N} (\partial_i \boldsymbol{\phi}_{j,\mathbf{W}}) \boldsymbol{\phi}_{j,\mathbf{W}}^\top\Big] W(\mathbf{W})^{-1}
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{\delta}(\theta)^\top \partial_i W(\mathbf{W})^{-1} \boldsymbol{\delta}(\theta) &= -2\boldsymbol{\delta}(\theta)^\top W(\mathbf{W})^{-1} \mathbb{E}\Big[\sum_{j=1}^{N} (\partial_i \boldsymbol{\phi}_{j,\mathbf{W}}) \boldsymbol{\phi}_{j,\mathbf{W}}^\top\Big] W(\mathbf{W})^{-1} \boldsymbol{\delta}(\theta) \\
&= -2\mathbf{u}(\mathbf{W})^\top \mathbb{E}\Big[\sum_{j=1}^{N} (\partial_i \boldsymbol{\phi}_{j,\mathbf{W}}) \boldsymbol{\phi}_{j,\mathbf{W}}^\top\Big] \mathbf{u}(\mathbf{W}) \\
&= -2\mathbf{u}(\mathbf{W})^\top \mathbb{E}\Big[\sum_{j=1}^{N} \boldsymbol{\phi}_{j,\mathbf{W}} (\partial_i \boldsymbol{\phi}_{j,\mathbf{W}})^\top\Big] \mathbf{u}(\mathbf{W})
\end{aligned}$$

The last line follows from the fact that the transpose of a scalar is equal to the scalar. Here the whole expression is transposed, leading to a transpose of the outer-product inside the sum. Additionally,

$$\begin{aligned}
\partial_i \boldsymbol{\delta}(\mathbf{W})^\top \mathbf{u}(\mathbf{W}) &= \mathbb{E}\Big[\sum_{j=1}^{N} \rho_j \boldsymbol{\delta}_j (\partial_i \boldsymbol{\phi}_{j,\mathbf{W}}) + \rho_j \boldsymbol{\phi}_{j,\mathbf{W}} \partial_i \boldsymbol{\delta}_j\Big]^\top \mathbf{u}(\mathbf{W}) \\
&= \mathbb{E}\Big[\sum_{j=1}^{N} \rho_j \boldsymbol{\delta}_j (\partial_i \boldsymbol{\phi}_{j,\mathbf{W}})^\top\Big] \mathbf{u}(\mathbf{W}) + \mathbb{E}\Big[\sum_{j=1}^{N} \rho_j \partial_i \boldsymbol{\delta}_j \boldsymbol{\phi}_{j,\mathbf{W}}^\top\Big] \mathbf{u}(\mathbf{W})
\end{aligned}$$

Grouping the terms with $(\partial_i \boldsymbol{\phi}_{j,\mathbf{W}})$,

$$\begin{aligned}
&\mathbb{E}\Big[\sum_{j=1}^{N} \rho_j \boldsymbol{\delta}_j (\partial_i \boldsymbol{\phi}_{j,\mathbf{W}})^\top\Big] \mathbf{u}(\mathbf{W}) - \mathbf{u}(\mathbf{W})^\top \mathbb{E}\Big[\sum_{j=1}^{N} \boldsymbol{\phi}_{j,\mathbf{W}} (\partial_i \boldsymbol{\phi}_{j,\mathbf{W}})^\top\Big] \mathbf{u}(\mathbf{W}) \\
&= \mathbb{E}\Big[\sum_{j=1}^{N} \Big(\rho_j \boldsymbol{\delta}_j - \mathbf{u}(\mathbf{W})^\top \boldsymbol{\phi}_{j,\mathbf{W}}\Big) (\partial_i \boldsymbol{\phi}_{j,\mathbf{W}})^\top \mathbf{u}(\mathbf{W})\Big] \\
&= \boldsymbol{\psi}_i(\mathbf{W})
\end{aligned}$$

where the last follows from the definition of $\nabla_{\mathbf{W}} \boldsymbol{\psi}(\mathbf{W})$, which is the gradient vector composed of partial derivatives $\boldsymbol{\psi}_i(\mathbf{W})$. Therefore,

$$\begin{aligned}\partial_i J(\mathbf{W}) &= 2\partial_i \boldsymbol{\delta}(\mathbf{W})^\top \mathbf{u}(\mathbf{W}) + \boldsymbol{\delta}(\theta)^\top \partial_i W(\mathbf{W})^{-1}\boldsymbol{\delta}(\theta)\\ &= 2\boldsymbol{\psi}_i(\mathbf{W}) + 2\mathbb{E}\left[\sum_{j=1}^{N} \rho_j \partial_i \boldsymbol{\delta}_j \boldsymbol{\phi}_{j,\mathbf{W}}^\top \mathbf{u}(\mathbf{W})\right]\end{aligned}$$

which proves Equation (5.17). Now the second term can be further simplified, using the fact that $\boldsymbol{\phi}_{j,\mathbf{W}} = \nabla_{\mathbf{W}} V_{j,\mathbf{W}}$, giving

$$\nabla_{\mathbf{W}} \boldsymbol{\delta}_j = \nabla_{\mathbf{W}} c_{j,\mathbf{W}} + \gamma_j \boldsymbol{\phi}'_{j,\mathbf{W}} - \boldsymbol{\phi}_{j,\mathbf{W}}.$$

Now notice that

$$\begin{aligned}\mathbb{E}\left[\sum_{j=1}^{N} \rho_j \nabla_{\mathbf{W}} \boldsymbol{\delta}_j \boldsymbol{\phi}_{j,\mathbf{W}}^\top \mathbf{u}(\mathbf{W})\right] &= -\mathbb{E}\left[\sum_{j=1}^{N} \rho_j \boldsymbol{\phi}_{j,\mathbf{W}} \boldsymbol{\phi}_{j,\mathbf{W}}^\top\right] \mathbf{u}(\mathbf{W})\\ &\quad + \mathbb{E}\left[\sum_{j=1}^{N} \rho_j \big(\nabla_{\mathbf{W}} c_{j,\mathbf{W}} + \gamma_j \boldsymbol{\phi}'_{j,\mathbf{W}}\big) \boldsymbol{\phi}_{j,\mathbf{W}}^\top \mathbf{u}(\mathbf{W})\right]\end{aligned}$$

Because $\mathbf{u}(\mathbf{W}) = W(\mathbf{W})^{-1}\boldsymbol{\delta}(\mathbf{W})$,

$$\mathbb{E}\left[\sum_{j=1}^{N} \rho_j \boldsymbol{\phi}_{j,\mathbf{W}} \boldsymbol{\phi}_{j,\mathbf{W}}^\top\right] \mathbf{u}(\mathbf{W}) = W(\mathbf{W})\mathbf{u}(\mathbf{W}) = \boldsymbol{\delta}(\mathbf{W})$$

Putting this all together,

$$\begin{aligned}-\tfrac{1}{2}\nabla_{\mathbf{W}} J(\mathbf{W}) &= -\boldsymbol{\psi}_i(\mathbf{W}) - \mathbb{E}\left[\sum_{j=1}^{N} \rho_j \nabla_{\mathbf{W}} \boldsymbol{\delta}_j \boldsymbol{\phi}_{j,\mathbf{W}}^\top \mathbf{u}(\mathbf{W})\right]\\ &= -\boldsymbol{\psi}(\mathbf{W}) + \boldsymbol{\delta}(\mathbf{W}) - \mathbb{E}\left[\sum_{j=1}^{N} \rho_j \big(\nabla_{\mathbf{W}} c_{j,\mathbf{W}} + \gamma_j \boldsymbol{\phi}'_{j,\mathbf{W}}\big) \boldsymbol{\phi}_{j,\mathbf{W}}^\top \mathbf{u}(\mathbf{W})\right]\end{aligned}$$

completing the proof. □

The resulting Recurrent GTD algorithm explicitly learns a second set of weights $\mathbf{u}$, to perform this update. In our implementation, a particular form of composition is used, namely that the cumulant for a GVF is a linear weighting of the predictions of some of the other GVFs on the next time-step. If $c(i,j)$ indicates the weight on the ith GVF in the cumulant for the jth GVF, then $\nabla_{\mathbf{W}} c_{j,t} = \sum_{i=1}^{N} c(j,i)\boldsymbol{\phi}'_{i,t}$.

The full **Recurrent GTD** update is

```
$\mathbf{s}_t \leftarrow f_{\mathbf{W}_t}(\mathbf{s}_{t-1}, \mathbf{x}_t)$
$\mathbf{s}_{t+1} \leftarrow f_{\mathbf{W}_t}(\mathbf{s}_t, \mathbf{x}_{t+1})$
$\boldsymbol{\phi}_{t,j} \leftarrow \nabla_{\mathbf{W}} \mathbf{s}_{t,j}$    $\triangleright$ Compute sensitivities using truncated BPTT
$\boldsymbol{\phi}'_{t,j} \leftarrow \nabla_{\mathbf{W}} \mathbf{s}_{t+1,j}$
$\rho_{t,j} \leftarrow \frac{\pi_j(a_t|\mathbf{o}_t)}{\mu(a_t|\mathbf{o}_t)}$
$\mathbf{v}_t = \nabla^2 \mathbf{s}_t \mathbf{u}_t$    $\triangleright$ Computed using R-operators, see Section 5.5
$\boldsymbol{\psi}_t = \sum_{j=1}^{N} (\rho_{j,t}\delta_{j,t} - \boldsymbol{\phi}_{j,t}^\top \mathbf{u}_t)\mathbf{v}_t$
$\mathbf{W}_{t+1} = \mathbf{W}_t + \alpha_t \left[ \sum_{j=1}^{N} \rho_{j,t}\boldsymbol{\delta}_{j,t}\boldsymbol{\phi}_{j,t} - \rho_{j,t}\left[\nabla_{\mathbf{W}} c_{j,t} + \gamma_{j,t+1}\boldsymbol{\phi}'_{j,t}\right]\boldsymbol{\phi}_{j,t}^\top \mathbf{u}_t - \boldsymbol{\psi}_t \right]$
$\mathbf{u}_{t+1} = \mathbf{u}_t + \beta_t \left[ \sum_{j=1}^{N} \rho_{j,t}\left(\boldsymbol{\delta}_{j,t} - \boldsymbol{\phi}_{j,t}^\top \mathbf{u}_t\right)\boldsymbol{\phi}_{j,t} \right]$
```

5.5 Computing Gradients of the Value Function Back Through Time

In this section, I show how to compute $\boldsymbol{\phi}_t$, which was needed in the algorithms. Recall from Section 5.3 that $V^{(j)}(\mathbf{s}_{t+1}) = \mathbf{s}_{t+1,j}$, and using $\mathbf{z}_{t+1} \overset{\text{def}}{=} \begin{bmatrix} \mathbf{s}_t \\ \mathbf{x}_{t+1} \end{bmatrix}$ let $\mathbf{s}_{t+1,j} = \sigma\left(\mathbf{z}_{t+1}^\top \mathbf{W}^{(j)}\right)$ for some activation function $\sigma : \mathbb{R} \to \mathbb{R}$. For both Backpropagation Through Time or Real Time Recurrent Learning, it is useful to take advantage of the following formula for *recurrent sensitivities*

$$\begin{aligned} \frac{\partial V^{(i)}(S_{t+1})}{\partial \mathbf{W}_{(k,j)}} &= \dot{\sigma}(\mathbf{z}_{t+1}^\top \mathbf{W}^{(i)}) \left(\left(\frac{\partial \mathbf{z}_{t+1}}{\partial \mathbf{W}_{(k,j)}} \right)^\top \mathbf{W}^{(i)} + (\mathbf{z}_{t+1})_j \delta^\kappa_{i,k} \right) \\ &= \dot{\sigma}(\mathbf{z}_{t+1}^\top \mathbf{W}^{(i)}) \left(\left[\frac{\partial V^{(1)}(S_t)}{\partial \mathbf{W}_{(k,j)}}, ..., \frac{\partial V^{(n)}(S_t)}{\partial \mathbf{W}_{(k,j)}}, \mathbf{0}^\top \right] \mathbf{W}^{(i)} + (\mathbf{z}_{t+1})_j \delta^\kappa_{i,k} \right) \end{aligned}$$

where δ^κ is the Kronecker delta function and $\dot{\sigma}(\cdot)$ is shorthand for the derivative of σ w.r.t its scalar input. Given this formula, BPTT or RTRL can simply be applied.

For Recurrent GTD—though not for Recurrent TD—Hessian needs to be computed back through time, for the Hessian-vector product. The Hessian for each value function is a $N((d+n)) \times N((d+n))$ matrix; computing the Hessian-vector product naively would cost at least $O(((d+n)+N)^2N^2)$ for each GVF, which is prohibitively expensive. This can be avoided using R-operators also known as Pearlmutter's method (Pearlmutter 1994).

The R-operator $\mathcal{R}\{\cdot\}$ is defined as

$$\mathcal{R}_{\mathbf{u}}\Big\{\mathbf{g}(\mathbf{W})\Big\} \overset{\text{def}}{=} \frac{\partial \mathbf{g}(\mathbf{W}+r\mathbf{u})}{\partial r}\bigg|_{r=0}$$

for a (vector-valued) function $\mathbf{g}$ and satisfies

$$\mathcal{R}_{\mathbf{u}}\Big\{\nabla_{\mathbf{W}} f(\mathbf{W})\Big\} = \nabla^2_{\mathbf{W}} f(\mathbf{W})\mathbf{u}.$$

Therefore, instead of computing the Hessian and then producting with $\mathbf{u}_t$, this operation can be completed in linear time, in the length of $\mathbf{u}_t$.

Specifically, for our setting,

$$\begin{aligned}
&\mathcal{R}_w\Big\{\dot{\sigma}(\mathbf{z}_t^\top\mathbf{W})[\nabla_{\mathbf{W}}\mathbf{z}_t^\top\mathbf{W} + \mathbf{z}_t^\top\nabla_{\mathbf{W}}\mathbf{W}]\Big\} \\
&\quad = \frac{\partial}{\partial r}\bigg(\dot{\sigma}(\mathbf{z}_t^\top(\mathbf{W}+r\mathbf{u})[\nabla_{\mathbf{W}}\mathbf{z}_t^\top(\mathbf{W}+r\mathbf{u}) + \mathbf{z}_t^\top\nabla_{\mathbf{W}}(\mathbf{W}+r\mathbf{u})]\bigg)\bigg|_{r=0}
\end{aligned}$$

To make the calculation more manageable each partial for every node k and associated weight j is separated.

$$\begin{aligned}
\frac{\partial V^{(i)}(S_{t+1},\mathbf{W})}{\partial \mathbf{W}_{(k,j)}} &= \dot{\sigma}(\mathbf{z}_{t+1}^\top\mathbf{W}^{(i)})(\eta_{t+1})_{i,k,j} \\
(\eta_{t+1})_{i,k,j} &= ((\mathbf{u}_t)_{k,j}^\top\mathbf{W}^{(i)} + (\mathbf{z}_{t+1})_j\delta_{i,k}) \\
(\mathbf{u}_t)_{k,j} &= \left[\frac{\partial V^{(1)}(S_t)}{\partial \mathbf{W}_{(k,j)}}, ..., \frac{\partial V^{(n)}(S_t)}{\partial \mathbf{W}_{(k,j)}}, \mathbf{0}^\top\right]^\top \\
\boldsymbol{\xi}_t &= \left[\frac{\partial V^{(1)}(S_t)}{\partial r}, ..., \frac{\partial V^{(n)}(S_t)}{\partial r}, \mathbf{0}^\top\right]^\top
\end{aligned}$$

$$
\begin{aligned}
(\mathcal{R}_t)_{w,V} &= \left[\mathcal{R}_w\left\{\frac{\partial V^{(1)}(S_{t-1})}{\partial \mathbf{W}_{(k,j)}}\right\}, ..., \mathcal{R}_w\left\{\frac{\partial V^{(N)}(S_{t-1})}{\partial \mathbf{W}_{(k,j)}}\right\}, \mathbf{0}^\top\right]^\top \\
\mathcal{R}_w\left\{\frac{\partial V^{(i)}(S_{t+1}, \mathbf{W})}{\partial \mathbf{W}_{(k,j)}}\right\} &= \left.\frac{\partial^2 V^{(i)}(S_{t+1}, \mathbf{W} + r\mathbf{u})}{\partial r \partial \mathbf{W}_{(k,j)}}\right|_{r=0} \\
&= \ddot{\sigma}\left(\mathbf{z}_{t+1}^\top \mathbf{W}^{(i)}\right)\left(\boldsymbol{\xi}_t^\top(\mathbf{W}^{(i)}) + \mathbf{z}_{t+1}^\top \mathbf{u}_i\right)(\eta_{t+1})_{i,k,j} \\
&\quad + \dot{\sigma}\left(\mathbf{z}_{t+1}^\top \mathbf{W}^{(i)}\right)\left((\mathcal{R}_t)_{w,V}^\top \mathbf{W}^{(i)} + (\mathbf{u}_t)_{k,j}^\top \mathbf{u}_i + (\boldsymbol{\xi}_t)_j \delta_{k,i}^\kappa\right) \\
\frac{\partial V^{(i)}(S_t)}{\partial r} &= \dot{\sigma}(\mathbf{z}_t^\top \mathbf{W}^{(i)})(\boldsymbol{\xi}_{t-1}^\top \mathbf{W}^{(i)} + \mathbf{z}_t^\top w_i)
\end{aligned}
$$

5.6 TD(λ) for GVFNs

For many of the experiments Recurrent TD is used with no back-propagation through time $p = 1$. This algorithm only adjusts parameters to minimize immediate TD error. In many cases, this was sufficient, but at times it was slow and increasing p improved learning. Another strategy is to use traces to obtain credit assignment back-in-time. The TD-error on this step can be attributed to state values back-in-time, with the **{**TD($\boldsymbol{\lambda}$) algorithm}

```
$\mathbf{s}_t \leftarrow f_{\mathbf{W}_t}(\mathbf{s}_{t-1}, \mathbf{x}_t)$
$\mathbf{s}_{t+1} \leftarrow f_{\mathbf{W}_t}(\mathbf{s}_t, \mathbf{x}_{t+1})$
$\mathbf{g}_{t,j} \leftarrow \nabla_{\mathbf{W}_j} f_{\mathbf{W}_t}(\mathbf{s}_{t-1}, \mathbf{x}_t)$        ▷ gradient given $\mathbf{s}_{t-1}$, no BPTT
$\mathbf{e}_{t,j} \leftarrow \mathbf{g}_{t,j} + \gamma_{t,j}\lambda \mathbf{e}_{t-1,j}$        ▷ eligibility trace, $0 \le \lambda \le 1$
$\delta_{t,j} \leftarrow C_{t+1}^{(j)} + \gamma_{t+1,j}\mathbf{s}_{t+1,j} - \mathbf{s}_{t,j}$
$\mathbf{W}_{t+1,j} \leftarrow \mathbf{W}_{t,j} + \alpha_t \boldsymbol{\delta}_{t,j}\mathbf{e}_{t,j}$
```

Notice the difference to Recurrent TD and Recurrent GTD, that the weights for each GVF are updated independently. This difference arises because the gradient computations for back-in-time, for the sensitivities, is what couples the updates. Without these sensitivities, the immediate gradient of the value $\mathbf{g}_{t,j}$ is independent for each GVF.

5.7 Summary

This chapter focused on the characterization of the mean squared projected bellman network error (MSPBNE) and subsequent derivations of algorithms based on this objective function. Specifically, I defined the Bellman network operator and the subsequent restrictions on the cumulants (i.e. acyclic composite connections). Equipped with the Bellman network operator and proving it is a contraction under some standard assumptions, the mean squared projected bellman network error can be defined. The final sections of the chapter derived the recurrent GTD and TD algorithms, and provided some insight into how to calculate various components with temporal sensitivities.

Chapter 6

Empirically Exploring Hand Designed GVFNs

In this chapter, I empirically explore GVFNs with hand-designed GVFs. These empirical investigations provide the initial evidence that GVFNs can learn without gradients calculated through BPTT. Through this empirical experimentation I show the importance of choosing GVFs as a means to develop predictive targets, explore the application of GVFNs on time-series forecasting, and finally show recurrent TD to be enough to learn in Ring World.

6.1 Experiments in Forecasting

In this section, I compare GVFNs and RNNs on two time series prediction datasets, particularly to ask 1) can GVFNs obtain comparable performance and 2) do GVFNs allow for faster learning, due to the regularizing effect of constraining the state to be predictions.[1] I investigate if they allow for faster learning both by examining learning speed as well as robustness to truncation length in BPTT.

6.1.1 Datasets

I consider two time series datasets previously studied in a comparative analysis of RNN architectures by Bianchi et al. 2017: the Mackey-Glass time series and the Multiple Superimposed Oscillator.

[1]All code for these experiments can be found at `https://github.com/mkschleg/GVFN`

The single-variate **Mackey-Glass (MG)** time series dataset is a synthetic data set generated from a time-delay differential equation:

$$\frac{\partial y(t)}{\partial t} = \alpha \frac{y(t-\tau)}{1+y(t-\tau)^{10}} - \beta y(t). \tag{6.1}$$

I follow the learning setup in Bianchi et al. 2017: setting $\tau = 17$, $\alpha = 0.2$, $\beta = 0.1$, and take integration steps of size 0.1. The target variable y is forcasted twelve steps into the future, starting from an initial value $y(0) = 1.2$. $T = 600,000$ samples are generated.

The **Multiple Superimposed Oscillator (MSO)** synthetic time series (Jaeger and Haas 2004) is defined by the sum of four sinusoids with unique frequencies

$$y(t) = \sin(0.2t) + \sin(0.311t) + \sin(0.42t) + \sin(0.51t). \tag{6.2}$$

The resulting oscillator has a long period of $2000\pi \approx 6283.19$. Because data is generated using $t \in \mathbb{N}$, the oscillator effectively never returns to a previously seen state. These attributes make prediction difficult with the MSO, as the model cannot rely on memory alone to make good predictions. $T = 600,000$ samples are generated and make predictions with a forecast horizon of $h = 12$.

6.1.2 Experimental Settings

The focus in this work is on online prediction, and so report online prediction error. At each step t, after observing $o_t = y(t)$, the RNN (or GVFN) makes a prediction $\hat{y}_t$ about the target y_t, which is the observation 12 steps into the future, $y_t = y(t+h)$. The magnitude of the squared error $(\hat{y}_t - y_t)^2$ depends on the scale of y_t. To provide a more scale invariant error, the error is normalized by the mean of the target—a mean predictor. Specifically, for each run, the average error over windows of size 10000 with the mean predictor is computed for each window is reported. This results in $T/10000$ normalized squared errors, where T is the length of the time series. This process is repeated 30 times, and average these errors across the 30 runs, and take the square root, to get a Normalized Root Mean Squared Error (NRMSE).

The values for hyperparameters are fixed as much as possible, using the previously reported value for the RNN and reasonable defaults for the GVFN. The stepsize is typically difficult to pick ahead of time, and so sweep that hyperparameter for all the algorithms. The number of hyperparameters swept are made comparable for all methods, to avoid an unfair advantage. The truncation length is not tuned, as results are reported for each truncation length $p \in \{1, 2, 4, 8, 16, 32\}$ for all the algorithms.

6.1.3 Algorithm Details

The GVFN consists of a single layer of size 32 and 128 (for MG and MSO respectively), corresponding to horizon GVFs. A horizon GVF is where the discount is constant value. As described in Section 4.4, each GVF has a constant continuation $\gamma^{(j)} \in [0.2, 0.95]$ and cumulant $C_t^{(j)} = \frac{1-\gamma^{(j)}}{y_t^{\max}} y(t)$, where $y_t^{\max}$ is an incrementally-computed maximum of the observations $y(t)$ up to time t. The GVFs are generated to linearly cover the range $[0.2, 0.95]$. This set is chosen as one of the simplest options that can be used without much domain knowledge. It is likely not the optimal set of GVFs for the GVFN, but represents a reasonable default choice. The GVFN is followed by a fully-connected layer with ReLU activations to produce a non-linear representation, which is linearly weighted to predict the target. The GVFN layer uses a linear activation, with clipping between [-10, 10], to help ensure state features remain bounded; again, this represented a simple rather than optimized choice.

The GVFN was trained using Recurrent TD with a constant learning rate and a batch size of 32. The weights for the fully-connected ReLU layer and the weights for the linear output are trained using ADAM, to minimize the mean squared error between the prediction at time t and target $y(t+h)$. The stepsize hyperparameters were swept: the learning rate for the GVFN $\alpha_{\text{GVFN}} = N \cdot 10^{-k}$ for $N \in \{1, 5\}$, $k \in \{3, \ldots, 6\}$, and the learning rate for the fully-connected and output layers $\alpha_{\text{pred}} = N \cdot 10^{-k}$ for $N \in \{1, 5\}$, $k \in \{2, \ldots, 5\}$.

I compare to RNNs, LSTMs, and GRUs [2]. The network architecture is similar to the GVFN for all recurrent models. The RNN size is set to 32 for

[2]Using standard implementations found in Flux (Innes 2018).

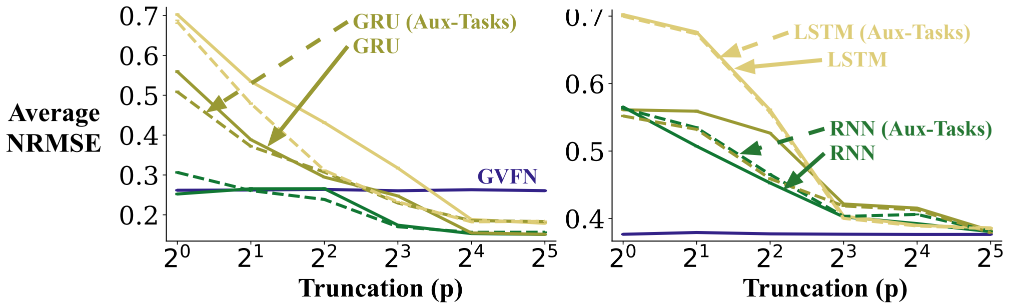


Figure 6.1: Truncation sensitivity for the (**left**) Mackey-Glass and (**right**) Multiple Superimposed Oscillator datasets. Errors are calculated using the normalized root mean squared error (NRMSE) averaged over the last 10k steps for the training results $\pm$ 1 standard error over 30 independent runs.

MG and 128 for MSO, while the GRU and LSTM have 8 hidden units for MG and 128 for MSO. Notice how the GRU and LSTM have fewer hidden units than the RNN and GVFN for the MG experiment. This roughly accounts for the increased complexity of the LSTMs and GRUs as compared to the GVFN and RNN. While this was needed to make all the models competitive in MG, I found the GVFNs performed well in MSO even with the same number of hidden units as the GRU and LSTMs. These models were trained using p-BPTT—specifically with the ADAM optimizer with a batch size of 32—to minimize the mean squared error between the prediction at time t and $y(t+h)$. The learning rate was swept in the range $\alpha = 2^{-k}$ with $k \in \{1, \ldots, 20\}$.

Finally, I also compare to RNNs with the 128 GVFs as auxiliary tasks. The augmented RNN has the same architecture as above, but with an additional set of output heads. The additional GVF heads are the same as those used by the GVFN, and are trained with TD. The gradient information from the GVFs is back-propagated through the network, influencing the representation. The augmented RNN was tuned over the same values as the RNN. The goal for adding this baseline is to gauge if there is an important difference in using the GVFs to directly constrain the state, as opposed to indirectly as auxiliary tasks. It further ensures that the RNN is given the same prior knowledge as the GVFN—namely the pertinence of these predictions—to avoid the inclusion of prior knowledge as a confounding factor.

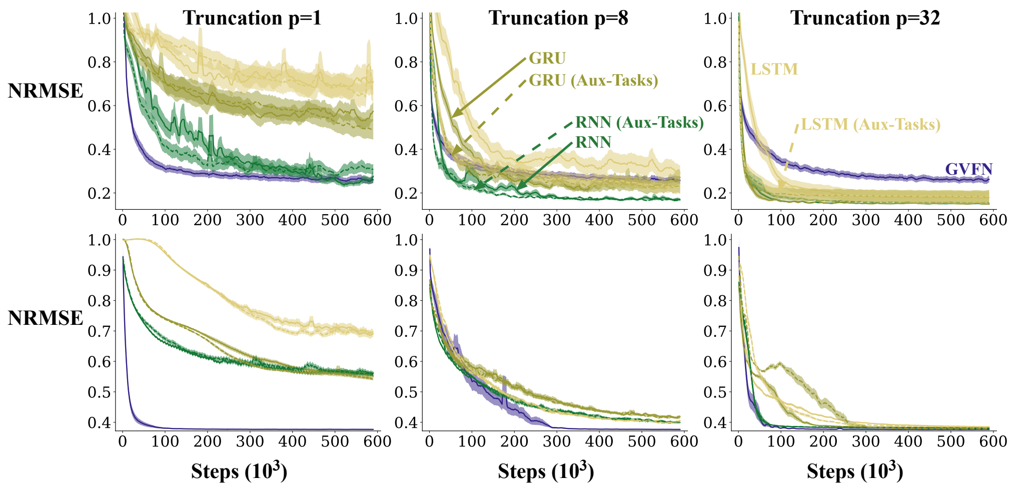


Figure 6.2: Learning curves for the (**top**) Mackey-Glass and (**bottom**) Multiple Superimposed Oscillator datasets. The normalized root mean squared error (NRMSE) normalized to the performance of the windowed average baseline is reported. All results are an average of 30 independent runs ± the standard error.

All RNNs and GVFNs include a bias unit, as part of the input as well as in all layers. All methods have similar computation per step, particularly as they are run with the same truncation levels p.

6.1.4 Results

I first show overall results across the truncation level in p-BPTT in Figure 6.1. Three results are consistent across both datasets: 1) GVFNs can obtain significantly better performance than RNNs with small p; 2) GVFNs are surprisingly robust to truncation level, providing almost the same performance across p; and 3) auxiliary tasks in the RNN do not provide consistent benefits across models and datasets. GVFNs provide a strict improvement on the MSO dataset. The result on MG is more nuanced. As truncation levels increase, the RNN's performance significantly improves and then passes the GVFN. This might suggest some bias in the specification of the GVFs. As is typical with regularization or imposing an inductive bias, it can improve learning—here allowing for much more stable learning with small p—but can prevent the solution from reaching the same prediction accuracy. In some cases, the inductive bias is strictly helpful, constraining the solution in the right way so as to incur minimal bias but improve learning. In MSO, it's possible the GVF specification was more appropriate and in MG less appropriate.

To gain more detailed insight into the behavior of the algorithms across truncation levels, learning curves for $p \in \{1, 8, 32\}$ are presented in Figure 6.2. All the approaches learn more slowly for $p = 1$, but the RNNs are clearly impacted more significantly. In MSO, the GVFN has a clear advantage in terms of learning speed. This is not true in MG, where once $p \geq 8$, the RNN performs better and learns faster. The GVFN objective here may actually be difficult to optimize, but it allows the agent to make progress constructing a useful state, whereas the signal from the error to the targets is insufficient.

6.2 Investigating Performance under Longer Temporal Dependencies

In this section, I investigate the utility of constraining states to be predictions, for an environment with long temporal dependencies. Compass World, introduced in Section 4.3 (see Figure 4.2), which can have long temporal dependencies, is used because the random behavior can stay in the center of the world for many steps, observing only the color white. The observation is encoded with two bits per color: one to indicate that color is observed, and the other to indicate another color is observed. The behavior policy chooses randomly between moving one-step forward; turning right/left for one step; moving forward until the wall is reached ({*leap*}); *or randomly selecting actions for k steps* ({wander}). The full observation vector is encoded based on which action was taken, and includes a bias unit.

Five hard-to-learn GVFs are used to evaluate the models. These GVFs correspond to the wall the agent is facing. These predictions are not learnable without constructing an internal state. These five questions correspond to leap questions. The leap question is defined as having a cumulant of 1 in the event of seeing a specific wall (orange, yellow, red, blue, green), and a continuation function defined as $\gamma = 0$ when any color is observed—when the agent is facing a wall—and $\gamma = 1$ otherwise.

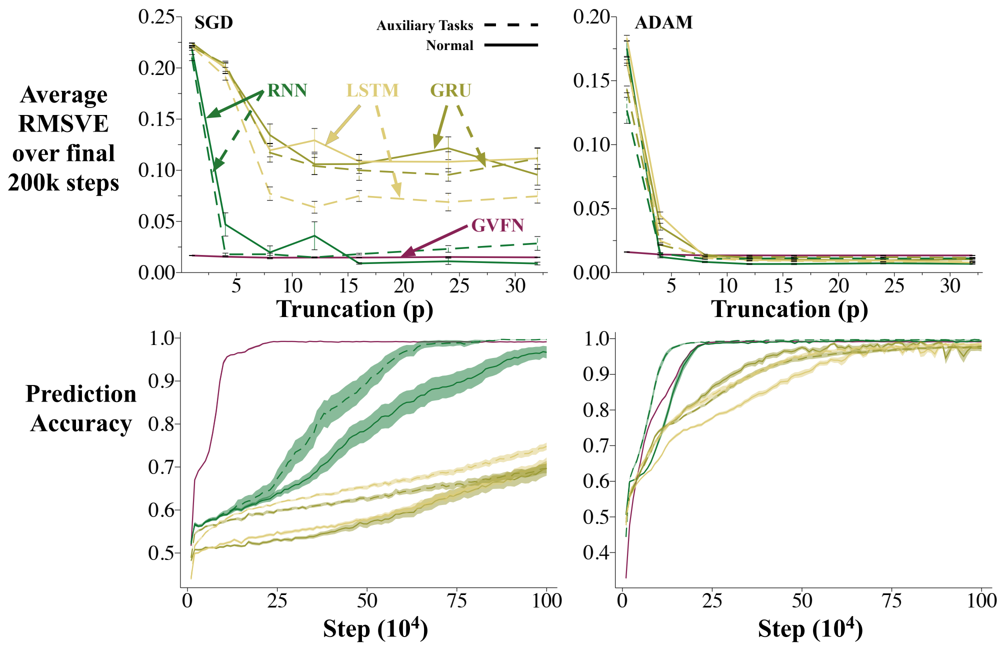


Figure 6.3: Results averaged over 30 runs ± one standard error. The dashed lines correspond to each RNN type augmented with auxiliary tasks, namely here the terminating horizon GVFs. The plots on the **(left)** are for a constant learning rate swept in range $\{0.1 \times 1.5^i; i \in [-10, 5]\} \cup \{1.0\}$. The plots on the **(right)** are for the ADAM optimizer with learning rate swept in range $\{0.01 \times 1.5^i; i \in \{-18, -16, \dots, 0\}\}$. The **(top)** row shows sensitivity over truncation measured by the average root mean squared value error (RMSVE) over the final 200000 steps of training. The **(bottom)** row shows learning curves for $p = 4$ for prediction accuracy. The prediction is the color of five walls with the highest GVF output, where the GVF prediction corresponds to a probability of facing that wall. When averaged over a window (10000 steps in our case) this results in a percentage of correct predictions during that time span.

The same architecture is used for both RNNs and GVFNs. The GVFN uses 40 GVFs: 8 GVFs per color. The 8 GVFs for a color correspond to **Terminating Horizon** GVFs. This means that they have a cumulant of 1 when seeing that color, and zero otherwise; they have a $\gamma = 1 - 2^k$ for one of 8 $k \in \{-7, -6, \dots, -1\}$; they terminate—$\gamma$ becomes zero—when any color is observed; and the policy is to always go forward. These GVFs are similar to the horizon GVFs in time series prediction, except that termination occurs when a wall is reached and the policy is off-policy. The RNN similarly uses 40 hidden units for the recurrent layer. For RNNs, the hyperbolic tangent is used. Sigmoids are used instead for GVFNs, because the returns are always nonnegative; otherwise, these two activations represent a similar architectural choice.

Treating the input action a_t specially significantly improved performance of both the RNN and GVFN, as suggested by Chapter 3. This is done by specifying separate weight vectors $\{w_a \in \mathbb{R}^n; \forall a \in \mathcal{A}\}$ for each action the agent can take. The hidden state is then calculated as $\mathbf{s}_{t+1} = \sigma(w_{a_t}^\top[\mathbf{o}_{t+1}, \mathbf{s}_t])$, where σ is the activation function. For the GRUs and LSTMs, this architectural modification is not straightforward; instead the action is passed in as a one-hot encoding.

All approaches share the same structure following the recurrent layer. The state $\mathbf{s}_t$ is passed to a 32-dimensional hidden layer with ReLU activation, and then is linearly weighted to produce the predictions for the five hard-to-learn GVFs: $\hat{\mathbf{y}}_t = \text{ReLU}(\mathbf{s}_t^\top \mathbf{F})\mathbf{W}$ where $\mathbf{F} \in \mathbb{R}^{40\times32}$ and $\mathbf{W} \in \mathbb{R}^{32\times5}$. All methods include a bias unit on every layer.

The performance for increasing p, as well as learning curves for $p = 8$, are show in Figure 6.3. Again, there are several clear conclusions. 1) The GVFN is again highly robust to truncation level, reaching almost perfect accuracy with $p = 1$. 2) The GVFN can learn noticeably faster with smaller p, such as $p = 4$, and the differences disappear for larger p. 3) The auxiliary tasks do not provide near the same level of benefit as the GVFN, though unlike the time series results, there does in fact seem to be some benefit. 4) All the methods are improved when using ADAM—especially the LSTMs and GRUs—though GVFNs are effective even with constant stepsizes.

6.3 Investigating Poorly Specified GVFNs

In the previous Compass World and Forecasting experiments, the GVFNs were robust to truncation. In fact, computing one-step gradients was sufficient for good performance. A natural question is when we can expect this to fail. I hypothesize that this robustness to truncation relies on appropriately specifying the GVFs in the GVFN. Poorly specified GVFs could both (a) make it so that the GVFN is incapable of constructing a state that can accurately predict the target and (b) make training difficult or unstable. In this section, this hypothesis is tested by evaluating several choices for the GVFs in the GVFN in Compass World.

Three additional GVFN specifications are considered: two that include intentional (but realistic) misspecifications and one that should be an improvement on the Terminating Horizon GVFN. The first misspecification, which is called the **Horizon** GVFN, causes the hidden states to have widely varying magnitudes. These GVFs are similar to the Terminating Horizon GVFs, except that they do not include termination when a color is observed. This means the true expected returns can be quite large, up to $\frac{1}{1-\gamma}$ (e.g., $\frac{1}{1-0.99} = 100$) if the agent is already immediately in front of the wall with that color. The policy is to go forward, and so if the agent is already facing the wall and receives a cumulant of 1, it will see a 1 forever onward, resulting in a return of $\sum_{i=0}^{\infty} \gamma^i = \frac{1}{1-\gamma}$.

The second misspecification provides a minimal set of sufficient predictions, but ones that are harder to learn. A natural choice for this is to use the five hard-to-learn predictions themselves, which is clearly sufficient but may be ineffective because they cannot be learned quickly enough to be a useful state. This is called the **Naive** GVFN, because it naively assumes that representability is enough, without considering learnability.

Finally, I also consider a specification that could improve on the more generic Terminating Horizon GVFN, that is called the **Expert Network**. This network also has 40 GVFs, but ones that are hand-designed for Compass World. This GVFN is a modified version of the TD network designed for Compass World (Sutton and Tanner 2005). The GVFs are defined similarly for the 5 colours. There are 3 myopic GVFs: a myopic GVFs consists of a myopic termination ($\gamma = 0$ always) and a cumulant of the color bit. Each myopic GVF has a persistent policy, which takes one action forever. Since there are three actions there are three myopic GVFs. These myopic GVFs indicate whether the agent is right beside the color (ahead, to the left or to the right). There is 1 leap GVF where the policy goes forward always, the cumulant is again

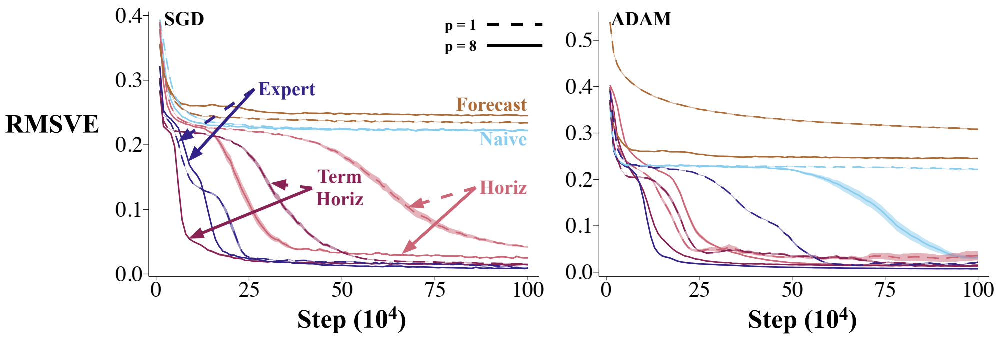


Figure 6.4: Learning curves for **(dashed)** $p = 1$ and **(solid)** $p = 8$ for various GVFN specifications and the Forecast networks. The GVFN is labeled TermHorizon, to highlight that it is composed of terminating horizon GVFs. Learning rates were chosen as in Figure 6.3, where the left plot corresponds to using a constant stepsize and the right to using the ADAM optimizer. The errors were averaged over 30 independent runs, to get the final learning curves $\pm$ standard error.

the color bit and $\gamma = 1$ except when a color is observed, giving $\gamma = 0$. There are 2 GVFs with a persistent policy (left, right) with myopic termination and a cumulant of the previous leap GVF's. These compositional GVFs let the agent know if they were to first turn right (or left) and then go forward, would they see the color. There are 2 leap GVFs with cumulants of the myopic GVFs. Finally, there is 1 GVF with uniform random policy with $\gamma = 0$ at a wall event and $\gamma = 0.5$ otherwise.

As a baseline, a **Forecast** network is included, which uses k-horizon predictions for the hidden state instead of GVFs. The architecture of the Forecast network is otherwise the same as the GVFN. The **Forecast** network uses a set of horizons $\mathcal{K} = \{1, 2, \ldots, 8\}$, for each of the non-white observations, resulting in a hidden state size of 40. To train these networks online a buffer is kept of $p + \max(\mathcal{K})$ observations, using the first p observations in the BPTT calculation and the next k observations to determine the targets of the network. The most recent hidden state is recovered to train the evaluation GVFs as done with the RNN and GVFN architectures. More specifically, at time-step t, the state $\mathbf{s}_{t-k}$ is updated with observations $\mathbf{o}_{t-k+1}, \ldots, \mathbf{o}_t$.

Learning curves for all the GVFN specifications, as well as the Forecast network, with $p = 1$ and $p = 8$, are reported in Figure 6.4. The results indicate that the specification can have a big impact. The two misspecified GVFNs perform noticeably worse than the Terminating Horizon GVFN. As expected, the Naive GVFN is eventually able to learn, with enough steps, $p = 8$ and the ADAM optimizer. It is sufficient to obtain a good state, but poor learnability prevents it from playing a useful role. The Horizon GVFN, which has potentially high magnitude GVF predictions, is closer in performance to the Terminating Horizon GVFN, but clearly worse. The Expert GVFN, on the other hand, can get to a lower error, though it does not have a clear advantage in terms of learning speed or robustness to p; this slower learning could again be potentially due to the fact that these expert GVFs were more difficult to learn than the simpler terminating horizon GVFs. Finally, the Forecast network performed very poorly. This is not too surprising in this environment. When considering a k-horizon prediction, the target is often zero, with the occasional one. This is generally a hard learning problem, as the resulting prediction loss does not provide a useful constraint. These results clearly show specifying the GVFs used to constrain the hidden state is an important consideration when using GVFNs, and could be the difference between learnable and un-learnable representations.

6.4 Comparing Recurrent GTD and Recurrent TD

TD networks with a simple TD network update rule—no backprop through time—have been shown to have divergence issues on a simple six-state domain, called Ring World~(Tanner and Sutton 2005). In fact, Gradient TD networks (Silver 2013) were introduced precisely to solve this problem. Because GVFNs are a strict generalization of TD networks, the GVFN can be set to get the same problematic setting if a simple TD update is used (RTD with $p = 1$). This raises a natural question of if Recurrent TD (RTD) similarly has divergence issues, and if Recurrent GTD (RGTD) is needed.

In all of the experiments so far, I have opted for the simpler RTD algorithm, rather than the full gradient algorithm RGTD, because empirically little

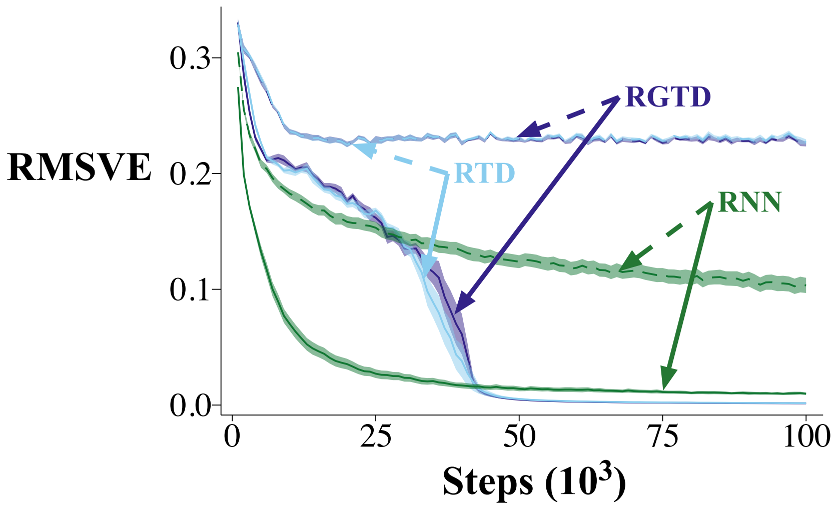


Figure 6.5: Learning curves for $p = 1$ and $p = 2$ averaged over 10 runs with fixed window smoothing of 1000 steps, in the Ring World environment. Learning rates chosen from a sweep over $\alpha \in \{0.1 \times 1.5^i; i \in \{-10, -9, \ldots, 6\}\}$ for the RNN and learning rates $\alpha \in \{0.1 \times 1.5^i; i \in \{-6, -9, \ldots, 8\}\}$ and $\beta = \{0.0, 0.01\}$ corresponding to RTD and RGTD respectively. All approaches needed only $p = 2$ to learn, including the baseline RNN included for comparison.

difference is found between the two. RTD, unlike the simple TD update rule, does in fact compute gradients back-in-time, and so should be a more sound update. Further, once truncated BPTT is used, even RGTD is providing a biased estimate of the gradient. But nonetheless RTD—which is built on the semi-gradient TD update—does drop more of the gradient than RGTD. It is likely that RGTD is needed in some cases. But it is possible that for most settings, RTD provides a reasonable interim choice between the simple TD network learning rule, and the more complex RGTD.

In this section, I test RTD and RGTD on Ring World, to see if they perform differently on this known problematic setting. Note that for $p = 1$, RTD reduces to the simple TD network learning rule, and so expect poor performance.

Ring World is a six-state domain~(Tanner and Sutton 2005) where the agent can move left or right in the ring. All the states are indistinguishable except state six. The observation vector is simply a two bit binary encoding indicating if the agent is in state six or not. The agent behaves uniformly randomly. The goal is to predict the observation bit on the next time-step. The environment itself is not too difficult for state-construction; rather a particular TD network causes divergence from the simple TD update rule. The corresponding GVFN consists of two chains of compositional GVFs: one chain for always go right and one chain for always going left. In the first chain, the first GVF is a myopic GVF, that has as cumulant the observed bit after taking action Right, with $\gamma = 0$. This first GVF predicts the observation one step into the future. The second GVF has the first GVFs prediction as a cumulant after taking action Right, with $\gamma = 0$. This second GVF predicts the observation two steps into the future. There are five GVFs in each chain, for a total of 10 GVFs in the GVFN.

Figure 6.5 shows the results of the Ring World experiments for truncation $p = 1$ and $p = 2$. The GVFNs for both RTD and RGTD needed only $p \geq 2$ to learn effectively. A baseline RNN of the same architecture is included, that indicates that the GVFN specification does negatively impact performance. But, with even just $p = 2$, any convergence issues seem to disappear. In fact, RTD and RGTD perform very similarly. The fact that Ring World is not problematic for RTD is by no means a proof that RTD is sufficient, especially since Ring World was designed to be a counterexample for the simple TD network update not for RTD. But, it is one more datapoint that RTD and RGTD perform similarly. In future work, counter examples should be investigated for RTD to better understand when it might be necessary to use RGTD.

6.5 Summary

This chapter contains some initial evidence to the effect of using GVFs to constrain the state of a recurrent network. While this provides some evidence for the *Prediction Representation hypothesis* defined in Section 4.2.1, I do not perform a explicit test of this hypothesis. While these experiments show the initial promise of the GVFN approach to learning a state-update function, they are not extensive. In all the empirical results, the agent's were trained in a sequential manner without experience replay buffers. This possibly limited the performance possible for both the RNNs and GVFNs. The results also relied on hand designed GVFs. While several networks used simple heuristics to construct a collection of GVFs, it is yet to be tested whether these results generalize beyond the simple bit domains and synthetic forecasting domains used here.

Chapter 7

The set of Predictive Questions in General Value Functions

In Chapter 6, I explored several hand designed network configurations. This lead to several observations, including compositional GVFs have considerable representational power if the predictive questions can be answered. In the following Chapter, I propose constraints we should apply to the set of GVFs. These constraints are born from the constraints on a continual learning agent, and inform future explorations in using GVFs and GVFNs for planning and control. I follow up this discussion by analyzing a specific type of GVF oft unexplored in the literature, composite GVFs. Specifically, I explore the open question of "what do GVFs composed together in chains predictively represent?" While I don't answer this question conclusively for all possible chains GVFs, I provide explorations for two important types of composite GVFs: those which have constant discounts, and those which have terminating discounts.

7.1 Composite GVFs

Consider the setting where there is an infinite sequence of sensor readings $\mathbf{x} = \{x[0], x[1], \dots, x[t], \dots, x[\infty]\}$ where $x[i] \in [x_{\min}, x_{\max}]$ and a constant discount γ. The return of this signal starting at a time-step t is $V[t] = \sum_{k=0}^{\infty} x[k]\gamma[k-t]$ where $\gamma[k] = \gamma^{k-1}$ for $k >= 1$ and 0 otherwise. This framing of the return is slightly different from the typical presentation. Specifically, the return can

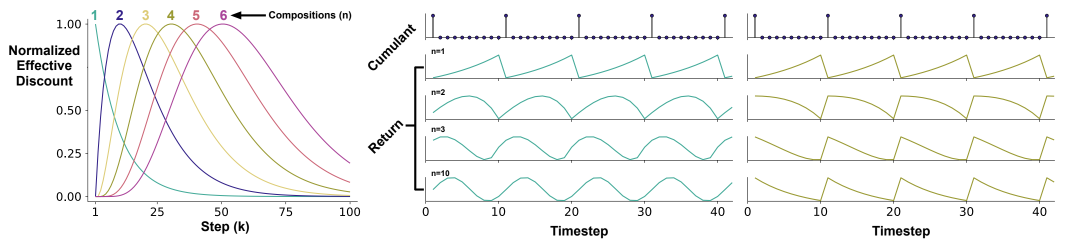


Figure 7.1: (**left**) The effective discount for n compositions normalized by the maximum value found in section 7.1. (**middle, right**) The Cycle World simulations, with top graph as the cumulant and subsequent plots n compositions with constant and terminating discounts respectively.

be interpreted as a convolution beteween γ and x [1] and shift the discount sequence over the sensor readings. This implicitly defines an infinite sequence of predictions $V[t]$. In the above equation, the sequence x is replaced with the sequence of predictions V getting a new set of predictions, and for any number of compositions

$$V^n[t] = \sum_{k=0}^{\infty} V^{n-1}[k]\gamma[k-t].$$

Expanding this equation the general sequence of effective discounts for n compositions and the corresponding return can be defined as

$$\gamma^n[k] = \begin{cases} 0 & \text{if } k < n \\ \frac{\prod_{i=1}^{n-1}(k-i)}{(n-1)!}\gamma^{k-n} \end{cases} \qquad V^n[t] = \sum_{k=0}^{\infty} x[k]\gamma^n[k-t]$$

where $\gamma^1[k] = \gamma[k]$ defined above and $V^n[t]$ is the target of the nth composition at time-step t. For any value n there are two sequences multiplied together. The original discounting shifted by the number of applications $\gamma^1[k-n]$ and a diverging series

$$Q^n[k] = \frac{\prod_{i=1}^{n-1}(k-i)}{(n-1)!} = \frac{\Gamma(k)}{\Gamma(k-n+1)\Gamma(n)}$$

where $\Gamma(k) = (k-1)!$ for $k \in \mathbb{Z}$ is known as the Gamma function, and can be used to analyze the function with $k \in \mathbb{R}$.

[1]In digital signal processing (Oppenheim and Schafer 2010) often the convolution, in this case γ, is mirrored across t and the inifinte sequence of sensor readings is $\mathbf{x} = \{x[-\infty], \ldots, x[t], \ldots, x[\infty]\}$. The corresponding convolution would be $V[t] = \sum_{k=-\infty}^{\infty} x[k]\gamma[t-k]$ which would change how the sequence of γ is defined. To be consistent with the reinforcement learning literature, γ is implicitly defined as the mirrored version and only consider the sequence starting at $k = 0$.

For any particular application of the convolution γ on a series with known domain $[x_{\min}, x_{\max}]$ the value function can take values bounded by $V^1[t] \in [\frac{x_{\min}}{1-\gamma}, \frac{x_{\max}}{1-\gamma}]$. This extends to n compositions in a straightforward way where the range of the value function becomes $V^n[t] \in [\frac{x_{\min}}{(1-\gamma)^n}, \frac{x_{\max}}{(1-\gamma)^n}]$. While normalizing the value function to take values in the range $[0, 1]$ has been used in various settings (Schlegel, Jacobsen, et al. 2021), as more compositions are added the effective range of values shrinks considerably.

Given the effective discounting sequence above, which observations are emphasized in the predictions can be uncovered. The first 100 steps of the effective discount function for several values of n can be seen in figure 7.1. These sequences are normalized to be in the range $[0, 1]$ for a visual comparison. The emphasis becomes increasingly spread as n increases, with the peak of this function moving further to the future at a consistent rate.

To find the maximum value, take the derivative of the log of the sequence with respect to k getting

$$\frac{\delta}{\delta k} \ln \gamma^n[k] = \psi(k) - \psi(k - n + 1) + \ln \gamma$$

where $\psi(z + 1) = H_z - C$ is the digamma function, $H_z = \sum_{i=1}^{z} \frac{1}{i} \leq \int_1^z \frac{1}{x} dx = ln(z)$ is the Euler harmonic number, and C is the Euler-Mascheroni constant. Using the approximation above, the maximal value is (to an approximation) $k = hn - (h - 1) = h(n - 1) + 1$, where $h = \frac{1}{1-\gamma}$ is sometimes known as the horizon of discount γ. Of course this is an approximation from above and the real value falls in $k \in [h(n - 1), h(n - 1) + 1)]$.

7.2 Empirical Observations

While one can describe the effective discount for composing constant discount predictions, the same techniques are difficult to apply to a non time-invariant discount (i.e. state-dependent discounts (Sutton, Modayil, et al. 2011; A. White 2015; M. White 2017)). Instead, in this section I look at the ideal returns of various signals using constant discounting and a terminating discounting functions. I use three datasets moving from highly synthetic to real-world

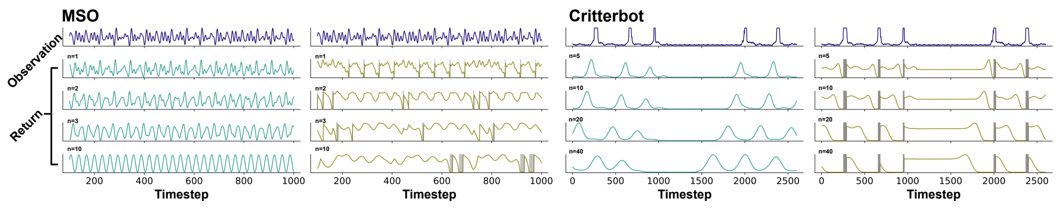


Figure 7.2: (**left two**) Returns of the multiple sinusoidal oscillator (MSO) synthetic dataset with constant and terminating discount respectively. The gray vertical lines are where the return terminates. (**right two**) Returns of Critterbot dataset over the light3 sensor with constant and terminating discount respectively.

robot sensori-motor data. The goal of this section is to show the non-intuitive behavior of compositions to motivate further analysis and exploration.[2] Below $\gamma = 0.9$ unless otherwise stated.

The first series is based off the cycle world, where the agent observes a sequence of a single active bit followed by 9 inactive bits, where the length of the sequence is $m = 10$. The cumulant is the observation itself, and in this section all learning is done through TD($\lambda = 0.9$) with learning rate $\alpha = 0.1$ and an underlying tabular representation where each component is the place in the sequence. Two chains of compositions were trained. The first is that of the continuous discounting described above, and the second is a series of discounts which terminate (i.e. $\gamma[t] = 0$) when the observation is active. The predictions of a single run can be seen in figure 7.1. For the constant discount, as the number of compositions increases the prediction sequence converge to what looks to be a sinusoid with frequency of 10, and amplitude driven by the analysis above. For the terminating discount, the wave form is more interesting. The first layer of predictions look very similar to the constant discount with amplitude shifted by $\frac{\gamma^m}{1-\gamma^m}$. But as there are more compositions the effect seems to be the prediction is at its height farther away from the active bit. As the agent gets closer to the observation, the sequence of summed values is shorter leading to smaller values. Given the sequence used, it is easy to mistake this as the agent creating a trace of the cumulant, but remember the prediction is about future cumulants.

[2]All code can be found at `https://github.com/mkschleg/CompGVFs.jl`

Next the Critterbot dataset (Modayil et al. 2014; A. White 2015) is used, focusing on light sensor 3 in figure 7.2. This gives a sequence of spikes similar to the cycle world sequence and a long pause in-between consistent saturations of the light sensor. The predictions look more like shifted and spread spikes as compared to the cycle world results. But with many more compositions, the return reverts to a similar form as before. The terminating discounts (with termination at sensor saturation $x[t+1] > 0.99$) provides a nice demonstration of how the returns are predicting the signal, just with a decaying prediction instead of the usual growing prediction. The results are similar in the multiple sinusoidal oscillator (Jaeger and Haas 2004). A slightly different terminating discount is used where the return terminates when the previous normalized prediction is $y^{n-1}[t+1] > 0.9$ rather than when the observation is saturated. While there are decays as the MSO sequence peaks, as the depth of the composition is increased, these periods are less frequent. Deep compositions may indicate parts of the sequence where there are fewer saturations in the original sequence.

7.3 Future Directions

This work suggests a number of interesting research directions and questions. While I mostly analyzed the sequence on discrete steps and applications of the filter, the general form does lend itself to continuous and complex values of n and k. In a similar vein, I focused on real valued exponential discounting while several discounting schemes exist which could be applied to our formulation. I am particularly interested in complex discounting (De Asis et al. 2018) and hyperbolic discounting (Ainslie and Haslam 1992; Fedus et al. 2019). Applying a diverse set of discounting schemes in compositions provide an interesting way to extend the power of value functions while maintaining learnability through efficient algorithms like temporal-difference learning.

The approach used in this chapter is unable to analyze state-dependent discount functions. One way around this might be in analyzing truncated sequences and taking an expectation over a distribution of sequence lengths.

This might lead to a expected effective discounting sequence, but how this will interact with an underlying Markov process is unclear. This is an important next step for understanding the effects of compositions in general value functions, and could also help in analyzing off-policy compositions.

Finally, the return can be re-interpreted as a convolution over the infinite sequence of observations. While this interpretation was only used to better the notation in this manuscript, further connections to convolutions and digital signal processing should be explored. Better filter designs might inspire different discounting schedules to squeeze more information from the data stream. I have also only analyzed these convolutions in the time domain. The frequency domain might give us more insight into how consistent signals like the cycle world dataset will be effected by compositions.

Chapter 8

Discovery of GVFNs through Generate and Test

In Chapter 6, I explored several hand designed network configurations. While a necessary first step to judge the GVFN's potential, discovering the predictive questions used to construct the state is essential to more aptly apply GVFNs to large complex problems. In chapter 7, I discussed the considerations needed when designing cumulant, discount, and policy functions for GVFs in a continual learning agent. In this chapter, I consider the discovery question for GVFNs in a continual learning setting. While this chapter only serves as the tip of the iceberg of what is possible with discovery, I aim to provide a baseline and structure to approach the question in future work. I also describe several approaches to discovery used throughout the predictive learning literature and discuss how they might apply to the GVFN architecture directly.

Previous approaches to discovery in predictive representations have focused on finding a set of predictions that would enable the agent to answer all predictive questions accurately. This objective is trying to find a sufficient statistic of the history for all predictions, and has been discussed in various forms (Subramanian et al. 2022). This is the approach typically taken in PSRs and a usual criteria when approaching a POMDP problem. This criteria falls naturally from the POMDP specification, where the assumption is there is a true underlying latent state which the agent can determine from enough interactions with the system. I conjecture that finding such a state is not feasible in large complex problems, and searching for such a state would be a poor use of a finite set of computational resources. Instead, the agent should focus on finding a set of questions which is useful for the agents overarching goals—for example, maximizing the return in the control problem. In the following section, I describe several prior approaches to discovery applicable to the GVFN framework, develop a simple approach to a discovery framework for future testing, and discuss various ways of specifying GVFs by hand for the GVFN.

8.1 Previous Approaches

There are two main families of approaches to discovery of GVFs for GVFNs: generate-and-test and gradient descent.

Generate and test is a natural algorithmic approach when considering a search problem through a complex unordered (or not obviously ordered) space. The core of the approach is to propose GVFs through a generator and approximate their utility for the downstream task through a proxy measure. This approach has been used for representation discovery (Mahmood and Sutton 2013; Javed 2020). The simplest setting where such a generate-and-test approach could be used is time series forecasting, as the predictions are on-policy and so policies do not have to be proposed by the generated. Further, practitioners can apply their prior knowledge in creating the cumulant and continuation functions considered by the generator. There are, however, some simple strategies for generating policies, which is discussed in Section 8.2.

A generate and test algorithm has been developed for TD networks (Makino and Takagi 2008). The process of discovery involves creating new predictions built entirely from existing structures: senses or predictions. By building new predictions from existing predictions, it facilitates the creation of composite structures. The system proposed in Makino and Takagi (2008) determines when a node (i.e. a prediction or sense) should be expanded on using three criteria. They then expand these nodes in specific ways to ask a broad set of composite questions. TD networks do not include policies—rather they include action primitives—so the approach does not directly extend. However, the idea of iteratively creating such composite structures does extend. For example, in this work, the expert network considered in Section 6.3 was composed of composite GVFs. Composite GVFs could be generated simply by using existing GVFs as the cumulant for the new GVF.

Meta-gradient descent uses gradient descent to learn meta-parameters that affect learning performance. The meta-parameters could correspond to initialization of a model for later fine tuning (Finn et al. 2017), a set of GVF auxiliary tasks to improve representation learning in Atari (Veeriah et al. 2019) or parameterized options (Bacon and Precup 2017). This approach splits the problem into two optimization problems: an inner problem and an outer problem. The inner optimization consists of the usual control or prediction procedure, where the agent seeks to maximize the discounted return or lower prediction error. The outer optimization calculates gradients through this procedure, with respect to the meta-parameters.

For example, to learn a set of GVFs as auxiliary tasks, Veeriah et al. (2019) parameterized the cumulant and continuation functions. They did not need to parameterize the policies for the GVFs as they assumed on-policy prediction: the policy π for the GVF is the current policy. These meta-parameters are optimized in the outer loop to produce auxiliary tasks that improve control performance in the inner loop. For this setting, one could similarly parameterize GVF questions, including the policy. This meta-gradient approach was reasonably effective for discovering GVFs as auxiliary tasks, though the procedure is expensive and has some trainability issues. Nonetheless, it is a reasonable direction for pursuing discovery for GVFNs.

8.2 Investigating a Simple Generate and Test Strategy for GVF Discovery

This simple discovery framework is based on algorithms described for representation search (Mahmood and Sutton 2013) focusing on two main components: an evaluator, and a generator. The evaluator is responsible for testing GVFs and removing unused GVFs. The generator proposes new GVFs from a set of possible GVFs. The framework is summarized in Figure 8.1. The key questions are how GVFs are evaluated and how new ones proposed. Our goal here is simply to demonstrate one avenue for discovery in GVFNs, rather than to develop an algorithm for discovery; this section opts for what are some of the simplest choices.

The magnitude of the associated weight in the external tasks using the GVFN is used to evaluate a GVF. The state vector is assumed to be used linearly to make predictions, with θ_j corresponding to state s_j and so to the jth GVF. All GVFs are evaluated every $N \in \mathbb{N}$ steps and prune the lowest $\epsilon \in [0, 1]$ percentage, i.e., prune $\lfloor n\epsilon \rfloor$ least useful GVFs of the full set of N GVFs. Other criteria have been proposed for evaluation, such as using traces of the weight magnitudes and considering internal weights (Mahmood and Sutton 2013). As mentioned above, the simplest choice that is still reasonably effective is opted for over more complex evaluation metrics.

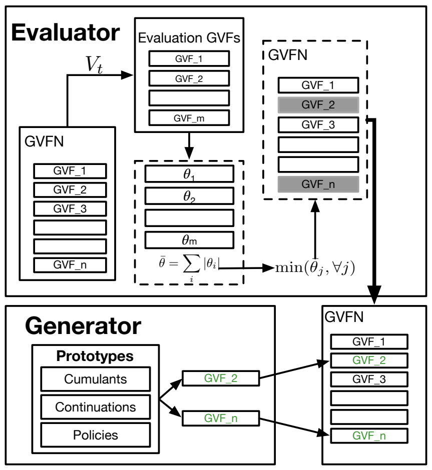


Figure 8.1: The discovery framework.

New GVFs are generated randomly from a set of GVF primitives. A set of basic types of cumulants, continuations and policies are randomly sampled from. For continuations, the set includes *myopic discounts* ($\gamma = 0$), *horizon discounts* ($\gamma \in (0, 1)$) and *terminating discounts* (the discount is set to $\gamma \in (0, 1]$ everywhere, except for at an event, which consists of a transition (o, a, o')). For cumulants, the set includes *stimuli cumulants* (the cumulant is one of the observations, or taking on 0 or 1 if the observation fulfills some criteria (e.g. a threshold)) and *composite cumulants* (the cumulant is the prediction of another GVF). *Random cumulants* (the cumulant is a random number generated from a zero-mean Gaussian with a random variance sampled from a uniform distribution) are also included; these are not expected to be useful, but rather use it to define what is called here a dysfunctional GVF to test pruning. For the policies, *random policies* (an action is chosen at random) and *persistent policies* (always follows one action) are used.

The resulting GVF primitives consist of a triplet (c, γ, π) where each is randomly chosen from these basic types. For example, a randomly generated GVF could consist of a myopic continuation, a stimuli cumulant on observation bit one and a random policy. This would correspond to predicting the first component of the observation vector on the next step, assuming a random action is taken. As another example, a randomly generated GVF could consist of a termination continuation with $\gamma = 0.9$, a stimuli cumulant which is 1 when the observation is zero and is otherwise zero otherwise and a persistent policy with action forward. This GVF corresponds to predicting the likelihood of seeing the observation change from active ('1') to inactive ('0'), given the agent persistently moves forward, within the horizon of about $(1 - \gamma)^{-1} = 10$ steps.

Parameterized continuations, cumulants and policies and randomly could have been considered. This set, however, is large. The GVF primitives can be seen as a prior over the full set of GVFs, which is too large from which to randomly generate. Without this prior the discovery approach is expected to still work but to take even longer than the experiments presented below.

The performance of the framework is evaluated on two experiments in Compass World (Sutton and Tanner 2005). Both experiments use the five hard-to-learn GVFs as the targets for the GVFN, introduced in Chapter 6. These questions correspond to a question of "which wall will I hit if I move forward forever?". The first experiment, Figure 8.2 (left), provides a sanity check that the evaluation strategy prunes dysfunctional representational units. The GVFN is initialized with 200 GVFs: 45 used to form the expert crafted TD network (Sutton and Tanner 2005), and 155 defective GVFs predicting noise $\sim \mathcal{N}(0, \sigma^2)$. In Figure 8.2 the learning curve and pruned GVFs over 12 million steps are reported. The second experiment, Figurere 8.2 (right), uses the full discovery approach to find a representation useful for learning the evaluation GVFs. The learning curves of the evaluative GVFs over 100 million steps are reported.

These experiments have many similarities to the experiments above, but

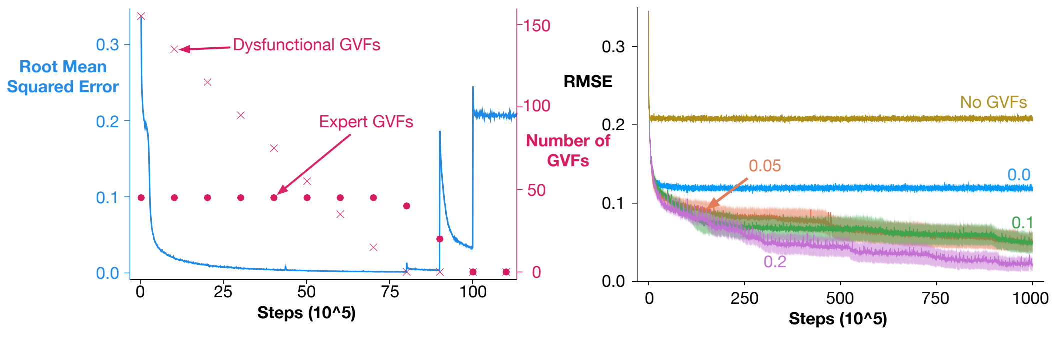


Figure 8.2: **(left)** Pruning predictive units occurs every million steps with no regeneration $\alpha = 0.001, \lambda = 0.9, \epsilon = 0.1, \sigma^2 = 1$ **(right)** Learning curves of the evaluative GVFs $N = 1000000, \epsilon =$ labeled, $\alpha = 0.001, \lambda = 0.9, n = 100$, over 5 runs with standard error denoted by the shaded region.

there is one key differences worth noting. Instead of using RTD or RGTD, TD(λ) is used; see Appendix 5.6 for the update equations. TD(λ) is sufficient to learn the expert network specification in a reasonable number of steps, and is significantly simpler than the other algorithms. Note that TD(λ) was not used in the above comparisons with RNNs. This was excluded for two reasons. First in the cases where the target was not a return, it is not possible to use eligibility traces, as they are designed for predicting expected returns. Second, as far as I am aware, the eligibility trace calculation for neural networks with several output nodes has not been formally derived nor tested.

The results indicate that even a simple generate and test approach can be effective for discovery in GVFNs. The first figure shows that the pruning approach gradually removes the dysfunctional GVFs, without pruning the expert GVFs. Eventually, once the agent has mostly removed all the dysfunctional GVFs, it is then forced to prune the expert GVFs and prediction performance begins to drop. Of course, in practice, the agent would not prune all its GVFs; in this experiment pruning is simply continued until the end. The second plot shows that iteratively pruning and generating new GVFs significantly improves on using an initial random set. For $\epsilon = 0.2$, which means about 20% of GVFs are pruned in each pruning phase, the prediction error continues to decrease until it almost reaches 0 and is almost as good as the set of hand-design GVFs used in previous experiments.

The goal of this experiment was to answer: is it possible to discover useful GVFs for a GVFN, even in simple settings? A negative answer would mean that GVFNs might have limited applicability. A demonstration that it is possible provides some evidence that this is a tractable problem for which even simple solutions can help us make traction. This demonstration, however, by no means shows an ideal or even efficient algorithm and there is ample room for improvement. Primarily, the random generation strategy does not take into account the current set of proposed predictions, potentially resulting in redundancy. A more principled method would look to generate a wide variety of predictions dependent on the current set of predictions.

8.3 Heuristics to Specify GVFNs

Through testing GVFNs in several domains there are some rules of thumb for choosing GVFs which can be applied today. In the time-series experiments, selecting GVFs with constant $\gamma^{(j)} \in [1-2^{-j}]$ where $j \in \mathbb{N}$ is surprisingly effective across the settings with fixed policies—namely the time series datasets. This is encouraging as these specifications on the surface seem simpler to discover than something as complex as the Expert network in Compass World. A set of discounts selected linearly across a range was also effective. Also including GVFs which have a pseudo-termination at a known event (known due to expert knowledge) and a cumulant which is only active at this event improved learning performance considerably (see the performance of the Terminating-Horizon network in Section 6.3).

Chapter 9

Importance Resampling for Prediction in Reinforcement Learning

One lingering issue that plagues not only GVFNs but the community of learning answers to predictive questions generally is the variance induced by off-policy prediction through importance sampling. There have been several alternatives to straight importance sampling ratios, many of which have been proposed for a more standard policy evaluation setting (i.e. where the target and behavior policies are close together on the simplex). In this Chapter, I propose importance resampling as an off-policy prediction algorithm to mitigate the update variance of importance sampling while also being a consistent estimator. I characterize the bias of this new estimator. Finally, I empirically investigate IR on three microworlds and a racing car simulator, learning from images, highlighting that (a) IR is less sensitive to learning rate than IS and V-trace (IS with clipping) and (b) IR converges more quickly in terms of the number of updates.

9.1 Introduction

An emerging direction for reinforcement learning systems is to learn many predictions, formalized as value function predictions contingent on many different policies. The idea is that such predictions can provide a powerful abstract model of the world. Some examples of systems that learn many value functions are the Horde architecture composed of General Value Functions (GVFs) (Sutton, Modayil, et al. 2011; Modayil et al. 2014), systems that use options (Sutton, Precup, et al. 1999; Schaul, Horgan, et al. 2015), predictive representation approaches (Sutton and Tanner 2005; Schaul and Ring 2013; Silver et al. 2017; Schlegel, Jacobsen, et al. 2021) and systems with auxiliary tasks (Jaderberg et al. 2017). Off-policy learning is critical for learning many value functions with different policies, because it enables data to be generated from one behavior policy to update the values for each target policy in parallel.

The typical strategy for off-policy learning is to reweight updates using importance sampling (IS). For a given state $\mathbf{s}$, with action a selected according to behavior μ, the IS ratio is the ratio between the probability of the action under the target policy π and the behavior: $\frac{\pi(a|\mathbf{s})}{\mu(a|\mathbf{s})}$. The update is multiplied by this ratio, adjusting the action probabilities so that the expectation of the update is as if the actions were sampled according to the target policy π. Though the IS estimator is unbiased and consistent (Kahn and Marshall 1953; Rubinstein and Kroese 2016), it can suffer from high or even infinite variance due to large magnitude IS ratios, in theory (Andradóttir et al. 1995) and in practice (Precup, Sutton, and Dasgupta 2001; Mahmood, van Hasselt, et al. 2014; Mahmood, Yu, et al. 2017).

There have been some attempts to modify off-policy prediction algorithms to mitigate this variance.[1] Weighted IS (WIS) algorithms have been introduced (Precup, Sutton, and Dasgupta 2001; Mahmood, van Hasselt, et al. 2014; Mahmood and Sutton 2015), which normalize each update by the sample average of the ratios. These algorithms improve learning over standard IS strategies, but are not straightforward to extend to nonlinear function approximation. In the offline setting, a reweighting scheme, called importance sampling with unequal support (Thomas and Brunskill 2017), was introduced to account for samples where the ratio is zero, in some cases significantly reducing variance. Another strategy is to rescale or truncate the IS ratios, as used by V-trace

[1]There is substantial literature on variance reduction for another area called off-policy policy evaluation, but which estimates only a single number or value for a policy (e.g., see Thomas and Brunskill 2016). The resulting algorithms differ substantially, and are not appropriate for learning the value function.

(Espeholt et al. 2018) for learning value functions and Tree-Backup (Precup, Sutton, and Singh 2000), Retrace (Munos et al. 2016) and ABQ (Mahmood, Yu, et al. 2017) for learning action-values. Truncation of IS-ratios in V-trace can incur significant bias, and this additional truncation parameter needs to be tuned.

An alternative to reweighting updates is to instead correct the distribution before updating the estimator using weighted bootstrap sampling: resampling a new set of data from the previously generated samples (Smith and Gelfand 1992; Arulampalam et al. 2002). Consider a setting where a buffer of data is stored, generated by a behavior policy. Samples for policy π can be obtained by resampling from this buffer, proportionally to $\frac{\pi(a|\mathbf{s})}{\mu(a|\mathbf{s})}$ for state-action pairs $(\mathbf{s}, a)$ in the buffer. In the sampling literature, this strategy has been proposed under the name Sampling Importance Resampling (SIR) (Rubin 1988; Smith and Gelfand 1992; Gordon et al. 1993), and has been particularly successful for Sequential Monte Carlo sampling (Gordon et al. 1993; Skare et al. 2003). Such resampling strategies have also been popular in classification, with over-sampling or under-sampling typically being preferred to weighted (cost-sensitive) updates (López et al. 2013).

A resampling strategy has several potential benefits for off-policy prediction.[2] Resampling could even have larger benefits for learning approaches, as compared to averaging or numerical integration problems, because updates accumulate in the weight vector and change the optimization trajectory of the weights. For example, very large importance sampling ratios could destabilize the weights. This problem does not occur for resampling, as instead the same transition will be resampled multiple times, spreading out a large magnitude update across multiple updates. On the other extreme, with small ratios, IS will waste updates on transitions with very small IS ratios. By correcting the distribution before updating, standard on-policy updates can be applied. The magnitude of the updates vary less—because updates are not multiplied by very small or very

[2]We explicitly use the term prediction rather than policy evaluation to make it clear that we are not learning value functions for control. Rather, our goal is to learn value functions solely for the sake of prediction.

large importance sampling ratios—potentially reducing variance of stochastic updates and simplifying learning rate selection. I hypothesize that resampling (a) learns in a fewer number of updates to the weights, because it focuses computation on samples that are likely under the target policy and (b) is less sensitive to learning parameters and target and behavior policy specification.

9.2 Background

This chapter considers the problem of learning GVFs (Sutton, Modayil, et al. 2011). The agent interacts in an environment defined by a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$ and Markov transition dynamics, with probability $\mathrm{P}(\mathbf{s}'|\mathbf{s}, a)$ of transitions to state $\mathbf{s}'$ when taking action a in state $\mathbf{s}$. A GVF is defined for policy $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$, cumulant $c : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and continuation function $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$, with $C_{t+1} \overset{\text{def}}{=} c(S_t, A_t, S_{t+1})$ and $\gamma_{t+1} \overset{\text{def}}{=} \gamma(S_t, A_t, S_{t+1})$ for a (random) transition (S_t, A_t, S_{t+1}). The value for a state $s \in \mathcal{S}$ is

$$V(\mathbf{s}) \overset{\text{def}}{=} \mathbb{E}_\pi \left[G_t | S_t = \mathbf{s}\right] \quad \text{where } G_t \overset{\text{def}}{=} C_{t+1} + \gamma_{t+1} C_{t+2} + \gamma_{t+1}\gamma_{t+2} C_{t+3} + \dots .$$

The operator $\mathbb{E}_\pi$ indicates an expectation with actions selected according to policy π. GVFs encompass standard value functions, where the cumulant is a reward. Otherwise, GVFs enable predictions about discounted sums of others signals into the future, when following a target policy π. These values are typically estimated using parametric function approximation, with weights $\theta \in \mathbb{R}^d$ defining approximate values $V_\theta(\mathbf{s})$.

In off-policy learning, transitions are sampled according to behavior policy, rather than the target policy. To get an unbiased sample of an update to the weights, the action probabilities need to be adjusted. Consider on-policy temporal difference (TD) learning, with update $\alpha_t \delta_t \nabla_\theta V_\theta(s)$ for a given $S_t = s$, for learning rate $\alpha_t \in \mathbb{R}^+$ and TD-error $\delta_t \overset{\text{def}}{=} C_{t+1} + \gamma_{t+1} V_\theta(S_{t+1}) - V_\theta(s)$. If actions are instead sampled according to a behavior policy $\mu : \mathcal{S} \times \mathcal{A} \to [0,1]$, then importance sampling (IS) to modify the update, giving the off-policy TD update $\alpha_t \rho_t \delta_t \nabla_\theta V_\theta(s)$ for IS ratio $\rho_t \overset{\text{def}}{=} \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$. Given state $S_t = \mathbf{s}$, if $\mu(a|s) > 0$ when $\pi(a|s) > 0$, then the expected value of these two updates are equal. To see why, notice that

$$\mathbb{E}_\mu\left[\alpha_t\rho_t\delta_t\nabla_\theta V_\theta(s)|S_t=s\right]=\alpha_t\nabla_\theta V_\theta(s)\mathbb{E}_\mu\left[\rho_t\delta_t|S_t=s\right]$$

which equals $\mathbb{E}_\pi\left[\alpha_t\rho_t\delta_t\nabla_\theta V_\theta(s)|S_t=s\right]$ because

$$\mathbb{E}_\mu\left[\rho_t\delta_t|S_t=\mathbf{s}\right]=\sum_{a\in\mathcal{A}}\mu(a|\mathbf{s})\frac{\pi(a|\mathbf{s})}{\mu(a|\mathbf{s})}\mathbb{E}\left[\delta_t|S_t=\mathbf{s},A_t=a\right]$$

Though unbiased, IS can be high-variance. A lower variance alternative is Weighted IS (WIS). For a batch consisting of transitions $\{(s_i,a_i,s_{i+1},c_{i+1},\rho_i)\}_{i=1}^n$, batch WIS uses a normalized estimate for the update. For example, an offline batch WIS TD algorithm, denoted WIS-Optimal below, would use update $\alpha_t\frac{\rho_t}{\sum_{i=1}^n\rho_i}\delta_t\nabla_\theta V_\theta(s)$. Obtaining an efficient WIS update is not straightforward, however, when learning online and has resulted in algorithms in the SGD setting (i.e. $n=1$) specialized to tabular (Precup, Sutton, and Dasgupta 2001) and linear functions (Mahmood, van Hasselt, et al. 2014; Mahmood and Sutton 2015). Nonetheless, WIS is used as a baseline in the experiments and theory.

9.3 Importance Resampling

In this section, Importance Resampling (IR) for off-policy prediction is introduced and its bias and variance is characterized. A resampling strategy requires a buffer of samples, from which to resample. Replaying experience from a buffer was introduced as a biologically plausible way to reuse old experience (Lin 1992; Lin 1993), and has become common for improving sample efficiency, particularly for control (Mnih et al. 2015; Schaul, Quan, et al. 2015). In the simplest case—which is assumed here—the buffer is a sliding window of the most recent n samples, $\{(s_i,a_i,s_{i+1},c_{i+1},\rho_i)\}_{i=t-n}^t$, at time-step $t>n$. Samples are generated by taking actions according to behavior μ. The transitions are generated with probability $d_\mu(s)\mu(a|s)\mathrm{P}(s'|s,a)$, where $d_\mu:\mathcal{S}\to[0,1]$ is the stationary distribution for policy μ. The goal is to obtain samples according to $d_\mu(s)\pi(a|s)\mathrm{P}(s'|s,a)$, as if actions were sampled according to policy π from states [3] $\mathbf{s}\sim d_\mu$.

[3]The assumption that states are sampled from d_μ underlies most off-policy learning algorithms. Only a few attempt to adjust probabilities d_μ to d_π, either by multiplying IS

The IR algorithm is simple: resample a mini-batch of size k on each step t from the buffer of size n, proportionally to ρ_i in the buffer. Using the resampled mini-batch the value function can be updated using standard on-policy approaches, such as on-policy TD or on-policy gradient TD. The key difference to IS and WIS is that the distribution itself is corrected, before the update, whereas IS and WIS correct the update itself. This small difference, however, can have larger ramifications practically, as shown in this chapter.

Two variants of IR are considered: with and without bias correction. For point i_j sampled from the buffer, let Δ_{i_j} be the on-policy update for that transition. For example, for TD, $\Delta_{i_j} = \delta_{i_j} \nabla_\theta V_\theta(s_{i_j})$. The first step for either variant is to sample a mini-batch of size k from the buffer, proportionally to ρ_i. Bias-Corrected IR (BC-IR) additionally pre-multiplies with the average ratio in the buffer $\bar{\rho} \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} \rho_i$, giving the following estimators for the update direction

$$X_{\text{IR}} \overset{\text{def}}{=} \frac{1}{k}\sum_{j=1}^{k} \Delta_{i_j} \qquad\qquad X_{\text{BC}} \overset{\text{def}}{=} \frac{\bar{\rho}}{k}\sum_{j=1}^{k} \Delta_{i_j}$$

BC-IR negates bias introduced by the average ratio in the buffer deviating significantly from the true mean. For reasonably large buffers, $\bar{\rho}$ will be close to 1 making IR and BC-IR have near-identical updates[4]. Nonetheless, they do have different theoretical properties, particularly for small buffer sizes n, so both are characterized.

Across most results, the following assumption is applied.

Assumption 4. *A buffer $B_t = \{X_{t+1}, ..., X_{t+n}\}$ is constructed from the most recent n transitions sampled by time $t + n$, which are generated sequentially from an irreducible, finite MDP with a fixed policy μ.*

ratios before a transition (Precup, Sutton, and Dasgupta 2001) or by directly estimating state distributions (Hallak and Mannor 2017; Liu et al. 2018). In this chapter, resampling is used to correct the action distribution—the standard setting. However, several insights are expected extend to how to use resampling to correct the state distribution, particularly because wherever IS ratios are used it should be straightforward to use our resampling approach.

In the sections below, I provide the bias characterizations of the two estimators defined above with proofs provided in the appendix. In the appendix, you can also find a characterization of the update variance.

[4] $\bar{\rho} \approx \mathbb{E}[\rho(a|s)] = \mathbb{E}[\frac{\pi(a|s)}{\mu(a|s)}] = \sum_{s,a} \frac{\pi(a|s)}{\mu(a|s)} \mu(a|s) d_\mu(s) = 1.$

To denote expectations under $p(x) = d_\mu(s)\mu(a|s)\mathrm{P}(s'|s,a)$ and $q(x) = d_\mu(s)\pi(a|s)\mathrm{P}(s'|s,a)$, the notation from above is overloaded, using operators $\mathbb{E}_\mu$ and $\mathbb{E}_\pi$ respectively. To reduce clutter, $\mathbb{E}$ is used to mean $\mathbb{E}_\mu$, because most expectations are under the sampling distribution.

9.3.1 Bias of IR

Below I show IR is biased, and that its bias is actually equal to WIS-Optimal, in Theorem 9.3.1

Theorem 9.3.1 (Bias for a fixed buffer of size n)**.** *Assume a buffer B of n transitions sampled i.i.d according to $p(x = (s,a,s')) = d_\mu(s)\mu(a|s)\mathrm{P}(s'|s,a)$. Let $X_{\mathrm{WIS}^*} \overset{\mathsf{def}}{=} \sum_{i=1}^n \frac{\rho_i}{\sum_{j=1}^n \rho_j}\Delta_i$ be the WIS-Optimal estimator of the update. Then,*

$$\mathbb{E}[X_{\mathrm{IR}}] = \mathbb{E}[X_{\mathrm{WIS}^*}]$$

and so the bias of X_{IR} is proportional to

$$\mathrm{Bias}(X_{\mathrm{IR}}) = \mathbb{E}[X_{\mathrm{IR}}] - \mathbb{E}_\pi[\Delta] \propto \frac{1}{n}(\mathbb{E}_\pi[\Delta]\sigma_\rho^2 - \sigma_{\rho,\Delta}\sigma_\rho\sigma_\Delta) \tag{9.1}$$

where $\mathbb{E}_\pi[\Delta]$ is the expected update across all transitions, with actions from S taken by the target policy π; $\sigma_\rho^2 = \mathrm{Var}(\frac{1}{n}\sum_{j=1}^n \rho_j)$; $\sigma_\Delta^2 = \mathrm{Var}(\frac{1}{n}\sum_{i=1}^n \rho_i\Delta_i)$; and covariance $\sigma_{(\rho,\Delta)} = \mathrm{Cov}(\frac{1}{n}\sum_{j=1}^n \rho_j, \frac{1}{n}\sum_{i=1}^n \rho_i\Delta_i)$.

Proof. Notice first that when we weight with ρ_i, this is equivalent to weighting with $\frac{d_\mu(S_i)\pi(A_i|S_i)\mathrm{P}(S_{i+1}|S_i,A_i)}{d_\mu(S_i)\mu(A_i|S_i)\mathrm{P}(S_{i+1}|S_i,A_i)}$, and so is the correct IS ratio for the transition.

$$\begin{aligned}
\mathbb{E}[X_{\mathrm{IR}}] &= \mathbb{E}\left[\mathbb{E}[X_{\mathrm{IR}}|B]\right] = \mathbb{E}\Big[\mathbb{E}\Big[\tfrac{1}{k}\sum_{j=1}^k \Delta_{i_j}|B\Big]\Big] \\
&= \mathbb{E}\Big[\tfrac{1}{k}\sum_{j=1}^k \mathbb{E}[\Delta_{i_j}|B]\Big] \qquad \triangleright\, \mathbb{E}[\Delta_{i_j}|B] = \sum_{i=1}^n \tfrac{\rho_i}{\sum_{j=1}^n \rho_j}\Delta_i \\
&= \mathbb{E}\Big[\sum_{i=1}^n \frac{\rho_i}{\sum_{j=1}^n \rho_j}\Delta_i\Big] \\
&= \mathbb{E}[X_{\mathrm{WIS}^*}].
\end{aligned}$$

This bias of X_{IR} is the same as X_{WIS^*}, which is characterized in Owen 2013, completing the proof. □

Theorem 9.3.1 is the only result which follows a different set of assumptions, primarily due to using the bias characterization of X_{WIS^*} found in Owen 2013. The bias of IR will be small for reasonably large n, both because it is proportional to $1/n$ and because larger n will result in lower variance of the average ratios and average update for the buffer in Equation (9.1). In particular, as n grows, these variances decay proportionally to n. Nonetheless, for smaller buffers, such bias could have an impact. The bias can be easily mitigated with a bias-correction term, as shown in the next corollary.

Corollary 9.3.1.1. *BC-IR is unbiased:* $\mathbb{E}[X_{\text{BC}}] = \mathbb{E}_\pi[\Delta]$.

Proof.

$$\begin{aligned}
\mathbb{E}[X_{\text{BC}}] &= \mathbb{E}\Big[\tfrac{\bar{\rho}}{k}\sum_{j=1}^{k}\mathbb{E}[\Delta_{i_j}|B]\Big] = \mathbb{E}\Big[\bar{\rho}\sum_{i=1}^{n}\frac{\rho_i}{\sum_{j=1}^{n}\rho_j}\Delta_i\Big] \\
&= \mathbb{E}\Big[\tfrac{1}{n}\sum_{i=1}^{n}\rho_i\Delta_i\Big] = \tfrac{1}{n}\sum_{i=1}^{n}\mathbb{E}\Big[\frac{\pi(A_i|S_i)}{\mu(A_i|S_i)}\Delta_i\Big] \\
&= \tfrac{1}{n}\sum_{i=1}^{n}\mathbb{E}\Big[\frac{d_\mu(S_i)\pi(A_i|S_i)\mathrm{P}(S_{i+1}|S_i,A_i)}{d_\mu(S_i)\mu(A_i|S_i)\mathrm{P}(S_{i+1}|S_i,A_i)}\Delta_i\Big] \\
&= \tfrac{1}{n}\sum_{i=1}^{n}\mathbb{E}_\pi\left[\Delta_i\right] = \mathbb{E}_\pi\left[\Delta\right].
\end{aligned}$$

The last equality follows from the fact that the samples are identically distributed. □

9.3.2 Consistency of IR

Consistency of IR in terms of an increasing buffer, with $n \to \infty$, is a relatively straightforward extension of prior results for SIR, with or without the bias correction, and from the derived bias of both estimators (see Appendix B.2 for proofs). More interesting, and reflective of practice, is consistency *with a fixed length buffer* and increasing interactions with the environment, $t \to \infty$. IR, without bias correction, is asymptotically biased in this case; in fact, its asymptotic bias is the one characterized above for a fixed length buffer in Theorem 9.3.1.

Theorem 9.3.2. *Let $B_t = \{X_{t+1}, ..., X_{t+n}\}$ be the buffer of the most recent n transitions sampled according to Assumption 4. Define the sliding-window estimator $X_t \overset{\text{def}}{=} \frac{1}{T}\sum_{t=1}^{T} X_{\text{BC}}^{(t)}$. Then, if $\mathbb{E}_\pi[|\Delta|] < \infty$, then X_T converges to $\mathbb{E}_\pi[\Delta]$ almost surely as $T \to \infty$.*

9.4 Weighted Importance Sampling

In this section, I detail the several variants of weighted importance sampling (WIS) used throughout the empirical evaluation (Section 9.5). All of these methods are either obvious variants of WIS when using an experience replay buffer in reinforcement learning or defined previously.

9.4.1 Mini-Batch Algorithms

Three weighted importance sampling updates are considered as competitors to IR. n is the size of the experience replay buffer, k is the size of a single batch. WIS-Minibatch and WIS-Buffer both follow a similar protocol as IS, in that they uniformly sample a mini-batch from the experience replay buffer and use this to update the value functions. The difference comes in the scaling of the update. The first, WIS-Minibatch, uses the sum of the importance weights ρ_i in the sampled mini-batch, while WIS-Buffer uses the sum of importance weights in the experience replay buffer. WIS-Buffer is also scaled by the size of the buffer and brought to the same effective scale as the other updates with $\frac{1}{k}$. WIS-Optimal follows a different approach and performs the best possible version of WIS where the gradient descent update is calculated from the whole experience replay buffer. Analysis on the bias or consistency of WIS-Minibatch or WIS-Buffer is not provided, but are natural versions of WIS one might try.

$$\Delta\theta = \frac{\sum_i^k \rho_i \delta_i \nabla_\theta V(s_i;\theta)}{\sum_j^k \rho_j} \qquad \text{WIS-Minibatch}$$

$$\Delta\theta = \frac{n}{k}\frac{\sum_i^k \rho_i \delta_i \nabla_\theta V(s_i;\theta)}{\sum_j^n \rho_j} \qquad \text{WIS-Buffer}$$

$$\Delta\theta = \frac{\sum_i^n \rho_i \delta_i \nabla_\theta V(s_i;\theta)}{\sum_j^n \rho_j} \qquad \text{WIS-Optimal}$$

9.4.2 Incremental Algorithm

While implementing an efficient true WIS algorithm for mini-batch updating is beyond the scope of this work, WIS-TD(0) is compared to the incremental versions of IR, IS, VTrace, and WISBatch. The difference between the mini-batch and incremental algorithms is how the updates are calculated. In the incremental scheme a random mini-batch of data is similarly sampled from the buffer. Each sample is used individually to update the value function. This is done to more naturally compare our baselines to WIS-TD(0) (Mahmood and Sutton 2015). WIS-TD(0) has parameters $u_0 \in \{\frac{1}{64}, 1, 5, 10, 50\} * 64, \mu \in 10^{-2:0.25:1}$, and $\eta = \frac{\mu}{u_0}$. WIS-TD(0) follows the update equations:

$$\begin{aligned}
\mathbf{u}_{i+1} &= (\mathbf{1} - \eta \boldsymbol{\phi}_i \circ \boldsymbol{\phi}_i) \circ \mathbf{u}_i + \rho_i \boldsymbol{\phi}_i \circ \boldsymbol{\phi}_i \quad \triangleright \circ \overset{\text{def}}{=} \text{ element wise product} \\
\alpha_{i+1} &= \mathbf{1} \oslash \mathbf{u}_{t+1} \quad \triangleright \oslash \overset{\text{def}}{=} \text{ element wise division} \\
\bar{\delta}_i &= C_i + \gamma_i \boldsymbol{\theta}_i^\top \boldsymbol{\phi}_i' - \boldsymbol{\theta}_{i-1}^\top \boldsymbol{\phi}_i \\
\boldsymbol{\theta}_{i+1} &= \boldsymbol{\theta}_t + \boldsymbol{\alpha}_{i+1} \circ \rho_i (\boldsymbol{\theta}_{i-1}^\top \boldsymbol{\phi}_i - \boldsymbol{\theta}_i^\top \boldsymbol{\phi}_i) \boldsymbol{\phi}_i + \rho_i \bar{\delta}_i \boldsymbol{\alpha}_{i+1} \circ \boldsymbol{\phi}_i
\end{aligned}$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ is the weight vector of the value function, $\boldsymbol{\phi}_i \in \mathbb{R}^d$ is the feature vector of the i-th transition in the experience replay buffer, and $\boldsymbol{\phi}_i' \in \mathbb{R}^d$ is the feature vector of the next state of the i-th transition in the experience replay buffer.

9.5 Empirical Results

Two hypothesized benefits of resampling as compared to reweighting are investigated: improved sample efficiency and reduced variance. These benefits are tested in two microworld domains—a Markov chain and the Four Rooms domain—where exhaustive experiments can be conducted. Finally, a demonstration that IR reduces sensitivity over IS and VTrace in a car simulator, TORCs, when learning from images is provided[5].

[5]Experimental code for every domain except Torcs can be found at https://mkschleg.github.io/Resampling.jl

IR and BC-IR are compared against several reweighting strategies, including importance sampling (IS); two online approaches to weighted important sampling, WIS-Minibatch with weighting $\rho_i / \sum_{j=1}^{k} \rho_j$ and WIS-Buffer with weighting $\rho_i / \frac{k}{n} \sum_{j=1}^{n} \rho_j$; and V-trace[6], which corresponds to clipping importance weights (Espeholt et al. 2018). WIS-TD(0) (Mahmood and Sutton 2015) is also used as a baseline, when applicable, which uses an online approximation to WIS, with a stepsize selection strategy (as described in Section 9.4.2). This algorithm uses only one sample at a time, rather than a mini-batch, and so is only included in Figure 9.2. Where appropriate, baselines using On-policy sampling are included; WIS-Optimal which uses the whole buffer to get an update; and Sarsa(0) which learns action-values—which does not require IS ratios—and then produces estimate $V(s) = \sum_a \pi(s, a)Q(s, a)$. WIS-Optimal is included as an optimal baseline, rather than as a competitor, as it estimates the update using the whole buffer on every step.

In all the experiments, the data is generated off-policy. The absolute value error (AVE) or the absolute return error (ARE) is computed on every step. For the sensitivity plots the average over all the interactions as specified for the environment are taken—resulting in MAVE and MARE respectively. The error bars represent the standard error over runs, which are featured on every plot—although not visible in some instances. For the microworlds, the true value function is found using dynamic programming with threshold 10^{-15}, and AVE is taken over all the states. For TORCs and continuous Four Rooms, the true value function is approximated using rollouts from a random subset of states generated when running the behavior policy μ, and the ARE is computed over this subset. For the Torcs domain, the same subset of states is used for each run due to computational constraints and report the mean squared return error (MSRE). Plots showing sensitivity over number of updates show results for complete experiments with updates evenly spread over all the interactions. A tabular representation is used in the microworld experiments,

[6]Retrace, ABQ and TreeBackup also use clipping to reduce variance. But, they are designed for learning action-values and for mitigating variance in eligibility traces. When trace parameter $\lambda = 0$—as assumed here—there are no IS ratios and these methods become equivalent to using Sarsa(0) for learning action-values.

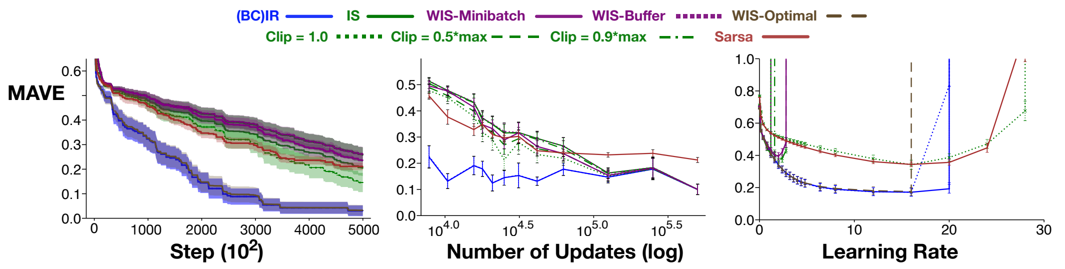


Figure 9.1: Four Rooms experiments ($n = 2500$, $k = 16$, 25 runs): **left** Learning curves for each method, with updates every 16 steps. IR and WIS-Optimal are overlapping. **center** Sensitivity over the number of interactions between updates. **right** Learning rate sensitivity plot.

tilecoded features with 64 tilings and 8 tiles is used in continuous Four Rooms, and a convolutional neural network is used for TORCs, with an architecture previously defined for self-driving cars (Bojarski et al. 2016).

9.5.1 Investigating Convergence Rate

The convergence rate of IR is investigated first. Learning curves and sensitivity to the learning rate are reported in the Four Rooms domain. The Four Rooms domain (Stolle and Precup 2002) has four rooms in an 11x11 grid world. The four rooms are positioned in a grid pattern with each room having two adjacent rooms. Each adjacent room is separated by a wall with a single connecting hallway. The target policy takes the down action deterministically. The cumulant for the value function is 1 when the agent hits a wall and 0 otherwise. The continuation function is $\gamma = 0.9$, with termination when the agent hits a wall. The resulting value function can be thought of as distance to the bottom wall. The behavior policy is uniform random everywhere except for 25 randomly selected states which take the action down with probability 0.05 with remaining probability split equally amongst the other actions. The choice of behavior and target policy induce high magnitude IS ratios.

As shown in Figure 9.1, IR has noticeable improvements over the reweighting strategies tested. The fact that IR resamples more important transitions from the replay buffer seems to significantly increase the learning speed. Further, IR has a wider range of usable learning rates. The same effect is seen even as the to-

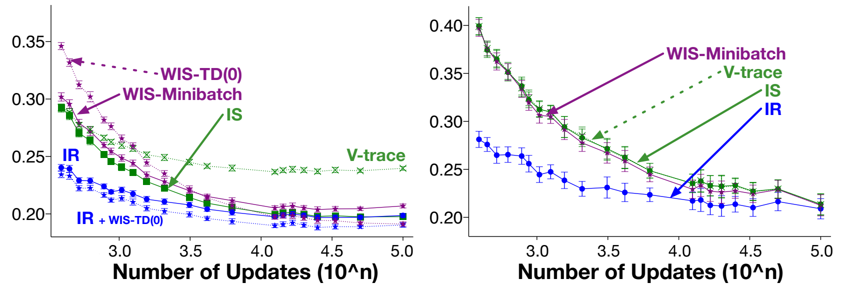


Figure 9.2: Convergence rates in Continuous Four Rooms averaged over 25 runs with 100000 interactions with the environment. **left** uniform random behavior policy and target policy which takes the down action with probability 0.9 and probability 0.1/3 for all other actions. Learning used incremental updates (as specified in section 9.4.2). **right** uniform random behavior and target policy with persistent down action selection learned with mini-batch updates with RMSProp.

tal number of updates is reduced, where the uniform sampling methods perform significantly worse as the interactions between updates increases—suggesting improved sample efficiency. WIS-Buffer performs almost equivalently to IS, because for reasonably size buffers, its normalization factor $\frac{1}{n}\sum_{j=1}^{n}\rho_j \approx 1$ because $\mathbb{E}[\rho] = 1$. WIS-Minibatch and V-trace both reduce the variance significantly, with their bias having only a limited impact on the final performance compared to IS. Even the most aggressive clipping parameter for V-trace—a clipping of 1.0— outperforms IS. The bias may have limited impact because the target policy is deterministic, and so only updates for exactly one action in a state. Sarsa—which is the same as Retrace(0)—performs similarly to the reweighting strategies.

The above results highlight the convergence rate improvements from IR, in terms of number of updates, without generalization across values. Conclusions might actually be different with function approximation, when updates for one state can be informative for others. For example, even if in one state the target policy differs significantly from the behavior policy, if they are similar in a related state, generalization could overcome effective sample size issues. Therefore, this phenomena is further investigated under function approximation with RMSProp learning rate selection.

Two experiments similar to above are conducted in a continuous state Four Rooms variant. The agent is a circle with radius 0.1, and the state consists of a continuous tuple containing the x and y coordinates of the agent's center point. The agent takes an action in one of the 4 cardinal directions moving $0.5 \pm \mathcal{U}(0.0, 0.1)$ in that directions with random drift in the orthogonal direction sampled from $\mathcal{N}(0.0, 0.01)$. The representation is a tile coded feature vector with 64 tilings and 8 tiles. Results are provided for both mini-batch updating (as above) and incremental updating (i.e. updating on each transition of a mini-batch incrementally, see appendix 9.4.2 for details). For the mini-batch experiment, the target policy deterministically takes the down action. For the incremental experiment, the target policy takes the down action with probability 0.9 and selects all other action with probability 0.1/3.

Generalization can mitigate some of the differences between IR and IS above in some settings, but in others the difference remains just as stark (see Figure 9.2 and Appendix B.3.2). If the behavior policy from the tabular domain, which skews the behavior in a sparse set of states, is used the nearby states mitigate this skew. However, if a behavior policy that selects all actions uniformly is used, then again IR obtains noticeable gains over IS and V-trace, for reducing the required number of updates, as shown in Figure 9.2.

Similar results are found for the incremental setting Figure 9.2 (left), where resampling still outperforms all other methods in terms of convergence rates. Given WIS-TD(0)'s significant degrade in performance as the number of updates decreases, I also compare with using WIS-TD(0) when sampling according to resampling IR+WIS-TD(0). Interestingly, this method outperforms all others — albeit only slightly against IR with constant learning rate. This result leads us to believe RMSProp may be a optimizer poor choice for this setting. Expanded results can be found in Appendix B.3.2.

9.5.2 Investigating Variance

To better investigate the update variance a Markov chain is used, this domain can more easily control dissimilarity between μ and π, and so control the magnitude of the IS ratios. The Markov chain is composed of 8 non-terminating

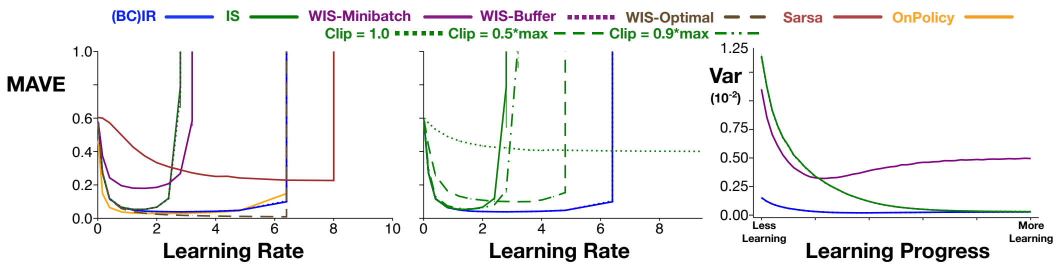


Figure 9.3: Learning Rate sensitivity plots in the Random Walk Markov Chain, with buffer size $n = 15000$ and mini-batch size $k = 16$. Averaged over 100 runs. The policies, written as [probability left, probability right] are $\mu = [0.9, 0.1], \pi = [0.1, 0.9]$ **left** learning rate sensitivity plot for all methods but V-trace. **center** learning rate sensitivity for V-trace with various clipping parameters **right** Variance study for IS, IR, and WISBatch. The x-axis corresponds to the training iteration, with variance reported for the weights at that iteration generated by WIS-Optimal. These plots show a correlation between the sensitivity to learning rate and magnitude of variance.

states and 2 terminating states on the ends of the chain, with a cumulant of 1 on the transition to the right-most terminal state and 0 everywhere else. The policies with probabilities [left, right] equal in all states: $\mu = [0.9, 0.1], \pi = [0.1, 0.9]$ are considered; further policy settings can be found in Appendix B.3.1.

First, the variance of the updates for fixed buffers is measured. I compute the variance of the update—from a given weight vector—by simulating the many possible updates that could have occurred. I am interested in the variance of updates both for early learning—when the weight vector is quite incorrect and updates are larger—and later learning. To obtain a sequence of such weight vectors, the sequence of weights generated by WIS-Optimal is used. As shown in Figure 9.3, the variance of IR is lower than IS, particularly in early learning, where the difference is stark. Once the weight vector has largely converged, the variance of IR and IS is comparable and near zero.

The update variance can be measured by proxy using learning rate sensitivity curves. As seen in Figure 9.3 (left) and (center), IR has the lowest sensitivity to learning rates, on-par with On-Policy sampling. IS has the highest sensitivity, along with WIS-Buffer and WIS-Minibatch. Various clipping parameters with V-trace are also tested. V-trace does provide some level of variance reduction but incurs more bias as the clipping becomes more aggressive.

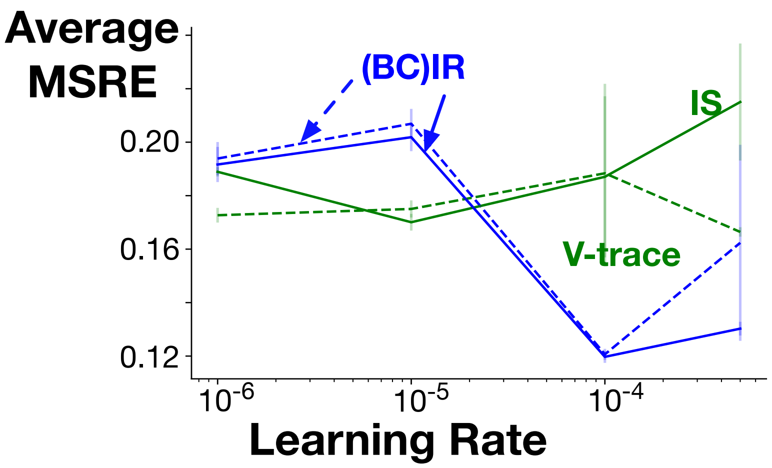


Figure 9.4: Learning rate sensitivity in TORCs, averaged over 10 runs. V-trace has clipping parameter 1.0. All the methods performed worse with a higher learning rate than shown here.

9.5.3 Demonstration on a Car Simulator

The TORCs racing car simulator is used to perform scaling experiments with neural networks to compare IR, IS, and V-trace. The simulator produces 64x128 cropped grayscale images. A deterministic steering controller is used. The controller produces steering actions $a_{det} \in [-1, +1]$ and take an action with probability defined by a Gaussian $a \sim \mathcal{N}(a_{det}, 0.1)$. The target policy is a Gaussian $\mathcal{N}(0.15, 0.0075)$, which corresponds to steering left. Pseudo-termination (i.e., $\gamma = 0$) occurs when the car nears the center of the road, and the cumulant becomes 1. Otherwise, the cumulant is zero and $\gamma = 0.9$. The policy is specified using continuous action distributions and results in IS ratios as high as ~ 1000 and highly variant updates for IS.

Again, IR provides benefits over IS and V-trace, in Figure 9.4. There is even more generalization from the neural network in this domain, than in Four Rooms where generalization did reduce some of the differences between IR and IS. Yet, IR still obtains the best performance, and avoids some of the variance seen in IS for two of the learning rates. Additionally, BC-IR actually performs differently here, having worse performance for the largest learning rate. This suggest IR has an effect in reducing variance.

9.6 Conclusions

In this Chapter, I introduced a new approach to off-policy learning: Importance Resampling. I showed that IR is consistent, and that the bias is the same as for Weighted Importance Sampling. I also provided an unbiased variant of IR, called Bias-Corrected IR. I empirically showed that IR (a) has lower learning rate sensitivity than IS and V-trace, which is IS with varying clipping thresholds; (b) the variance of updates for IR are much lower in early learning than IS and (c) IR converges faster than IS and other competitors, in terms of the number of updates. These results confirm the theory presented for IR, which states that variance of updates for IR are lower than IS in two settings, one being an early learning setting. Such lower variance also explains why IR can converge faster in terms of number of updates, for a given buffer of data.

The algorithm and results in this Chapter suggest new directions for off-policy prediction, particularly for faster convergence. Resampling is promising for scaling to learning many value functions in parallel, because many fewer updates can be made for each value function. A natural next step is a demonstration of IR, in such a parallel prediction system. Resampling from a buffer also opens up questions about how to further focus updates. One such option is using an intermediate sampling policy. Another option is including prioritization based on error, such as was done for control with prioritized sweeping (Peng and R. Williams 1993) and prioritized replay (Schaul, Quan, et al. 2015).

Chapter 10

Conclusions and Final Remarks

This thesis set out to uncover some of the key discrepancies between how recurrent models work in the supervised learning setting and the continual reinforcement learning setting. In this chapter, I will summarize the main contributions made to agent-state construction in the continual RL setting. I will discuss potential future directions and propose open questions and directions for developing methods of agent-state construction based on the work presented in this thesis.

10.1 Summary of Contributions

The work presented in this thesis takes a deep look into using recurrent networks in the reinforcement learning setting. I focus on two assumptions brought over from supervised learning and construct an architecture to take advantage of temporal-difference learning to learn agent state. Through controlled experimentation and a focused approach the contributions which make up this thesis show several limitations of recurrent networks as used in the supervised learning setting when applying them to the reinforcement learning and continual reinforcement learning settings. Some of these limitations were alleviated by leveraging ideas inspired by predictive representations of state (Littman and Sutton 2002; Tanner 2005), with several future research directions identified.

First, the incorporation of actions and other information was investigated in the DRQN framework in Chapter 3. In this chapter, I discussed several approaches to incorporate information into the cell of a recurrent network found throughout the literature. I incorporated these approaches into both simple recurrent cells and gated-recurrent units to uncover the advantages of each. I showed that, in most cases, we should be applying the action through a multiplicative update, and that combining architectures can be used to find the appropriate cell for a problem.

Next, I developed a predictive approach to learning state which restricts the agent-state to predictions of the observation stream. Approaching this question resulted in a new predictive framework for agent-state construction known as *General Value Function Networks* (Chapter 4). I explored the similarities of this approach to previous predictive approaches, and showed GVFNs encompasses the predictive targets of several previous directions.

To fully develop the predictive approach found in Chapter 4, I defined a set of learning updates following in the footsteps of gradient temporal-difference algorithms and previous predictive architectures in 5. These algorithms follow naturally from the gradient temporal-difference learning updates (Maei, Szepesvári, Bhatnagar, Silver, et al. 2009; Sutton, Maei, et al. 2009) and gradient temporal-difference networks (Silver 2013). Our learning update extends beyond these approaches to incorporate composite predictive questions into the objective as well as arbitrary discounting (with the usual limitations (M. White 2017)).

In Chapter 6, I empirically compared the usual approaches to agent-state construction in supervised learning with our predictive approach in several continual reinforcement learning and continual timeseries domains. I showed the incorporation of predictions which can be learned through temporal-difference learning results in a recurrent network which can learn absent of gradients rolled back through time. These predictions were also competitive to predictions made by other recurrent architectures when allowed large truncations in BPTT. I investigated several predictive questions which result in a wide range of performance and also highlight GVFs learned through temporal-difference as a key component. Finally, I reproduced a known counter example for temporal-different networks in ring world and showed this can be solved through the use of recurrent semi-gradient temporal difference learning rather than the full gradient update rule derived in section 4.

The empirical results discussed in 6 lead to questions about the underlying targets presented by composite predictive questions and what an algorithm to discover structures like this may look like. I posed and provided initial explanations and explorations in Chapters 7 and 8 respectively. These results create a promising future research direction in developing new methods for discovering predictive questions and combing GVFs to create novel hand designed structures which might be relevant across a wide range of problems

Finally, posing off-policy prediction questions for learning an agent-state requires us to re-think the core approach to off-policy prediction. Due to traditional importance sampling can quickly result in large variance unstable updates, I proposed importance resampling as a technique to avoid the instability induced by large importance sampling ratios in Chapter 9. This new approach to off-policy prediction effectively manages the update variance found when using importance sampling ratios directly in the update, while also improving the efficiency of each update towards learning the target prediction.

In this thesis, I not only contributed to our understanding and art in learning agent-state for the partial observable reinforcement learning setting, I developed a general purpose approach to learning agent-state through off-policy temporal-difference learning and showed the utility of such an approach as compared to the usual approach found in supervised learning. This thesis lays the ground work for a wide range of interesting research questions about learning agent-state. More details of these future directions can be found below.

10.2 Future Directions and Work

In this section, I outline several promising future research directions derived from the work presented in this thesis.

10.2.1 Open Problems for Recurrent Architectures in RL

Recurrent architectures are often taken off-the-shelf from the supervised learning setting for use in reinforcement learning. While this has been moderately successful, the RL problem poses challenges not often considered by supervised learning. Below I discuss three interesting properties of an RL system, and how they affect learning using recurrent networks.

Practical Online Recurrent Learning:

In reinforcement learning, it is desirable to learn as much about the most recent experience before selecting an action (i.e. to learn online and incrementally). Learning efficiently online enables adapting behavior in real time and scaling to massive data-streams and architectures. This puts pressure on the learning system to update the weights within a set amount of time so the system can act (Sutton, Modayil, et al. 2011; A. White 2015), which is often not a concern in the supervised setting. In settings where an agent must move around its environment independently, the on-board computational system can be heavily constrained by the power of the processor as well as limited energy from the battery. An algorithm whose computational and memory complexity scales independently of the sequence length (without the quadratic complexity on size of the network as real time recurrent learning (R. J. Williams and Zipser 1989)) and could be applied online-incrementally would be a major breakthrough in using recurrent architectures for RL and computationally constrained systems generally.

Active Data Collection Matters:

Imagine an agent in a hallway with recognizable observations only at the beginning and end of the hallway, much like our TMaze environments. The agent must learn a state update which spans at least the length of the hallway. But this is in the best case scenario when the agent prioritizes making it to the end of the hallway. In reality, our agents will randomly explore the hallway until the end, often extending the length of the sequence the agent needs to learn over. The interaction between the agent's behavior (or exploration) and the difficulty of training under partial observability with a recurrent agent is currently unexplored. Active data collection strategies could mitigate the length of long-temporal dependencies, which would show massive improvements in our agent's learning efficiency and ability.

To show the potential of active data collection, I will briefly revisit the

Directional TMaze. Start with an agent who has learned the base task of the Directional TMaze environment. If the agent is forced to take specific actions and to start in a specific orientation at the beginning of the episode, we could feasibly teach the agent to artificially extend the horizon of its policy without increasing the length of the training sequence. See Figure 10.1 for preliminary results of how behavior can extend the horizon of the agent's policy. In this experiment, trained multiplicative agents are paced through a set of interventions. The naive strategy uses epsilon greedy after forcing the agent to step forward twice down the hallway. The hand designed sequence, instead guides the agent through a series of forced actions to build to the final desired policy. This simple experiment shows the potential for slowly extending the temporal horizon of a policy without adjusting the truncation value by intervening on the agent's behavior.

Insight Beyond Learning Curves:

Learning curves provide little understanding of an agent's learning process and this likely limits algorithmic progress in partially observable settings. Unfortunately, such metrics can't be used to address more complicated questions about the agent and its behavior. While searching for SOTA is admirable, deeper questions about the internal learned structures and behaviors of our agents are often, but not always, ignored. Analyzing the internal dynamics of an agent with recurrent architectures is uniquely challenging in reinforcement learning. Some challenges include:

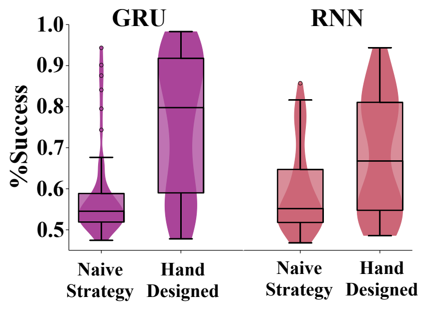


Figure 10.1: Average success over the intervention taking the go forward activation and starting in the eastward position. **(Naive Strategy)** Using the evaluated intervention over 60k steps for training, **(Hand Designed)** a sequence of hand designed interventions to build up to the final evaluation intervention over 60k steps.

- Generating data for evaluating and analyzing representation learning is an especially difficult problem for agents with recurrence. The data generated must be coherent trajectories the agent may potentially experience in the environment, meaning a (or several) data generating policy must be selected to provide coverage over the space of agent-environment interactions

- Current tools for analyzing state representations are designed for NLP (Karpathy et al. 2015; Ming et al. 2017) and are ill-suited for analyzing the link between the environment, the agent's state, and the behavior policy.

- Analyzing the behavior of our agents through performance metrics leaves many questions unanswered: How the agent might be behaving? When does the agent make a long-term decision? In what circumstances might the agent's policy fail?

Learning curves showing the agent's performance, usually through episodic return or prediction error, over the agent's lifetime has been the primary method algorithms are compared. Unfortunately, such metrics can't be used to address more complicated questions about the agent and its behavior. While searching for SOTA is admirable, deeper questions about the internal learned structures and behaviors of our agents are often, but not always, ignored. Analyzing the internal dynamics of an agent with recurrent architectures is uniquely challenging in reinforcement learning.

One challenge is how data is generated. Unlike SL whose data is usually a dataset designed ahead of time, RL generates data through interactions with an environment whose underlying dynamics are likely unaccessible to the system designer. While randomly generated data, in combination with tools from NLP (Karpathy et al. 2015; Ming et al. 2017), can give us some insight into how our agent's perform, see section 3.5, extending the analysis to larger domains could leave large parts of the agent-environment interactions unseen.

While I provide some representational analysis in the prediction setting, further results in the control setting would be even more beneficial. Unfortunately, many of the analysis tools considered require the "correct" target at a given time. In the control setting, even when the underlying dynamics of an environment can be fully specified (say in lunar lander) a notion of what the right action is at a given time can be extremely difficult to discover. Future work should go into analyzing the link between histories, agent state, environment state, and behavior.

Even when analyzing the behavior of our agent, using the performance metric as the primary measure is deeply flawed. This type of analysis fails to address questions such as: How the agent might be behaving? When does the agent make a long-term decision? In what circumstances might the agent's policy fail? Analyzing the agent as a non-linear coupled system with the environment through a series of dynamical equations could lead to further insight on conditions which lead the agent to behave in certain ways or when certain decisions are made by the agent. Beer 2003 develop a series of questions and experiments to analyze an artificial agent from this perspective in a simple catcher like domain. While these tools would be difficult to apply to real-world problems, using them in simulations could provide useful insight into a full description of the agent's learned policy.

Because of the above challenges there are several lingering questions about these types of agents left unanswered in these domains. 1) Under what conditions will the agent's policy fail in an environment? 2) How robust are the policies to out-of-distribution events and how does this effect the hidden state? 3) What algorithms do the learning process discover to solve the domains reliably? 4) Is the model stable over a long training period or in a continual domain? 5) When does the agent make a decision, and does the agent stick to this decision? I believe answering these questions and more can lead to better understanding of recurrent agents as well as pathways to better algorithms for training such agents.

10.2.2 Learning How to Encode Action in Recurrent Architecture through Experience

So far, we've focused on architectures which have static architectures, where the agent has no agency in learning the appropriate structure. While this strategy seems to be reasonable as a starting point, in the future an architecture which

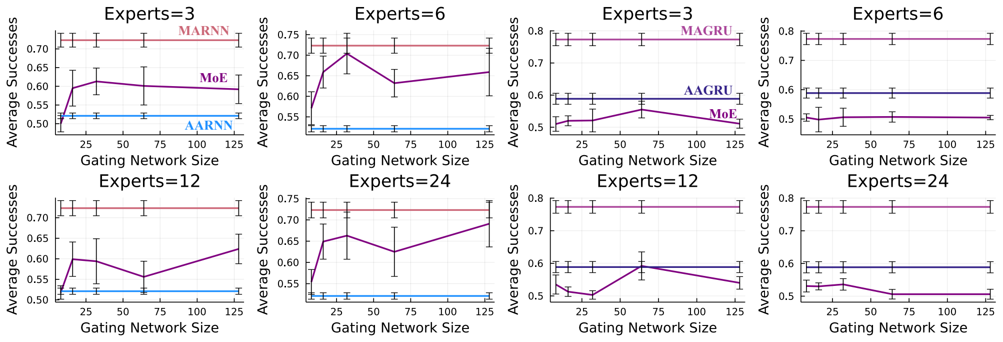


Figure 10.2: Directional TMaze sweep over size of the gating network (i.e how many units in a single hidden layer with ReLU activations and an output network w/ softmax output) for (**left**) RNN and (**right**) GRU. All experiments follow the procedures from previous results, except the MoE networks use 20 runs.

can learn these different networks would be more desirable. I propose one such architecture here and an initial empirical evaluation of this architecture, leading to a discussion on the problematic properties such an algorithm might have.

$$\begin{aligned} z_t^i &= f_{\text{update}}(\mathbf{W}, \mathbf{x}_t, \mathbf{a}_{t-1}) \\ \psi_t &= f_{\text{GN}}(\mathbf{x}_t, \mathbf{a}_{t-1}) \\ \mathbf{s}_t &= z_t \odot \psi_t. \end{aligned}$$

Where $f_{\text{GN}} : \mathcal{A} \times \mathcal{S} \times \mathcal{O} \to \mathbb{R}^{|\mathbf{s}|}$ is a parameterized function which is used to create a mixture over the experts state $z \in \mathbb{R}^{|\mathbf{s}|}$ produced by a state update function f_{update}. Both the gating function and the expert rnn state update function can be arbitrarily constructed. In this section, I focus on the simple RNN update and a feed forward ANN with ReLU activations and a softmax activation on the final layer.

The results are presented in Figure 10.2. A sweep over various number of experts and a simple gating network with a single layer and softmax activation. As compared to the additive and multiplicative the mixture of experts RNN network performs in-between the two networks. The GRU, on the other hand, fails to perform well in this domain. This might be related to the results seen in section 3.7, where both the combined GRUs failed to outperform the multiplicative.

10.2.3 Understanding the Intersection between Predictive and Non-Predictive Agent-State Construction

While this thesis creates a dichotomy between predictive and non-predictive approaches to learning agent-state, this is generally not a necessary distinction. One key question is to combine the two approaches into a network that both takes advantage of temporal-difference for learning state as well as has the flexibility to learn state objects which aren't dictated by predictive questions. Such an architecture could improve on the sensitivity to truncation for many recurrent networks while also providing a depth of representation not available in the current state of the GVFN framework. Future work should go into understanding such an architecture.

10.2.4 Discovery and the Form of GVF Prediction Targets

As explored in Chapters 7 and 8, little is known about the shape of composite predictive targets or of the space of predictive questions generally. while several works appeal to the ability of GVFs to specify questions about everything the agent could imagine, we must be careful that prior knowledge in the space of questions an agent can specify is avoided. Instead, questions should be built on the sequence of observations from the environment. This can be done through a meta-learning process or generate and test (see Chapter 8), but little effort has gone into understanding the complex structures of these predictive question networks.

10.2.5 Beyond a Synchronous Predictive Network

Like co-agent networks before them, the GVFN framework provides a mechanism for de-linking the final objective of the agent and its internal representation learning. Using this framework to derive a asynchronous set of architectures could provide a lower-variance scheme as compared to the co-agent network work. The key reason this is possible is because the targets of the individual units are now predictions of the underlying data stream (or any other available signal), meaning the variance in predictions should be the direct result of the underlying uncertainty or variance in the actual data stream. This is unlike co-agent networks as they rely on random sampling for each co-agent's activations. An approach which combines prediction and control demons in the GVFN framework could be key to creating such an asynchronous network.

10.3 Conclusion

This thesis investigated how off-policy prediction could be applied to improve state construction in reinforcement learning. The general value function network architecture shows off-policy prediction can be leveraged to improve state construction in RL when using recurrent networks in several ways. Through these investigations we developed several learning algorithms and future directions in taking advantage of off-policy prediction for state construction. I also discussed many of the implications of these choices, and thought about what the limits of this approach may be in both how expressive GVFs are and in the methods for discovering sets of GVFs for a GVFN.

This thesis also provides evidence that the intuitions developed about recurrent neural networks in supervised learning might not directly apply to the continual reinforcement learning setting. While many of the approaches derived in SL could be useful in RL, these should be re-investigated, validated, and understood in the continual reinforcement learning setting. While I propose several directions for improvement in recurrent networks for CRL, this thesis lays the groundwork for many avenues to contribute to the use and understanding of recurrent networks in CRL. The future of recurrent networks CRL will likely share many similarities with those found in SL, but will likely have radically different approaches to construction, learning, and analysis which cannot be developed in the SL setting alone.

Bibliography

Ainslie, George and Nick Haslam (1992). “Hyperbolic Discounting.” In: *Choice over Time*. New York, NY, US: Russell Sage Foundation.

Andradóttir, Sigrún, Daniel P. Heyman, and Teunis J. Ott (1995). “On the Choice of Alternative Measures in Importance Sampling with Markov Chains.” In: *Operations Research*.

Arulampalam, M. S., S. Maskell, N. Gordon, and T. Clapp (2002). “A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking.” In: *IEEE Transactions on Signal Processing*.

Bacon, Pierre-Luc and Doina Precup (2017). “The Option-Critic Architecture.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Baird, Leemon (1995). “Residual Algorithms: Reinforcement Learning with Function Approximation.” In: *Machine Learning Proceedings 1995*. Ed. by Armand Prieditis and Stuart Russell. San Francisco (CA): Morgan Kaufmann.

Bakker, Bram (2002). “Reinforcement Learning with Long Short-Term Memory.” In: *Advances in Neural Information Processing Systems 14*. MIT Press.

Banino, Andrea, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J. Chadwick, Thomas Degris, Joseph Modayil, Greg Wayne, Hubert Soyer, Fabio Viola, Brian Zhang, Ross Goroshin, Neil Rabinowitz, Razvan Pascanu, Charlie Beattie, Stig Petersen, Amir Sadik, Stephen Gaffney, Helen King, Koray Kavukcuoglu, Demis Hassabis, Raia Hadsell, and Dharshan Kumaran (2018). “Vector-Based Navigation Using Grid-like Representations in Artificial Agents.” In: *Nature*.

Barreto, André, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, Hado van Hasselt, and David Silver (2018). “Successor Features for Transfer in Reinforcement Learning.” In: *arXiv:1606.05312 [cs]*. arXiv: `1606.05312 [cs]`.

Baum, Leonard E. and Ted Petrie (1966). “Statistical Inference for Probabilistic Functions of Finite State Markov Chains.” In: *The Annals of Mathematical Statistics*. JSTOR: `2238772`.

Becker, Joseph D (1973). “A Model for the Encoding of Experiential Information.” In: *Computer models of thought and language*.

Beer, Randall D. (2003). “The Dynamics of Active Categorical Perception in an Evolved Model Agent.” In: *Adaptive Behavior*.

Bengio, Yoshua, N. Boulanger-Lewandowski, and R. Pascanu (2013). “Advances in Optimizing Recurrent Networks.” In: *IEEE International Conference on Acoustics, Speech and Signal Processing*.

Bianchi, Filippo Maria, Enrico Maiorino, Michael C. Kampffmeyer, Antonello Rizzi, and Robert Jenssen (2017). *Recurrent Neural Networks for Short-Term Load Forecasting: An Overview and Comparative Analysis*. SpringerBriefs in Computer Science. Cham: Springer International Publishing.

Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba (2016). *End to End Learning for Self-Driving Cars*. arXiv: `1604.07316 [cs]`.

Boots, Byron, Sajid M Siddiqi, and Geoffrey J Gordon (2011). “Closing the Learning-Planning Loop with Predictive State Representations.” In: *The International Journal of Robotics Research*.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). “Openai Gym.” In: *arXiv:1606.01540*. arXiv: `1606.01540`.

Bubic, Andreja, D. Yves Von Cramon, and Ricarda I. Schubotz (2010). “Prediction, Cognition and the Brain.” In: *Frontiers in Human Neuroscience*.

Butz, Martin V., Olivier Sigaud, and Pierre Gérard (2003). “Anticipatory Behavior: Exploiting Knowledge About the Future to Improve Current Behavior.” In: *Anticipatory Behavior in Adaptive Learning Systems: Foundations, Theories, and Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg.

Chandar, Sarath, Chinnadhurai Sankar, Eugene Vorontsov, Samira Ebrahimi Kahou, and Yoshua Bengio (2019). “Towards Non-Saturating Recurrent Units for Modelling Long-Term Dependencies.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Chen, Lili, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch (2021). “Decision Transformer: Reinforcement Learning via Sequence Modeling.” In: *Advances in Neural Information Processing Systems 34*.

Cho, Kyunghyun, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio (2014). “On the Properties of Neural Machine Translation: Encoder–Decoder Approaches.” In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Association for Computational Linguistics.

Choromanski, Krzysztof, Carlton Downey, and Byron Boots (2018). “Initialization Matters: Orthogonal Predictive State Recurrent Neural Networks.” In: *International Conference on Learning Representations*. PMLR.

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio (2014). “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.” In: *arXiv:1412.3555 [cs]*. arXiv: `1412.3555 [cs]`.

Clark, Andy (2013). "Whatever next? Predictive Brains, Situated Agents, and the Future of Cognitive Science." In: *Behavioral and Brain Sciences*.

Cunningham, Michael (2013). *Intelligence: Its Organization and Development*. Elsevier.

Dai, Bo, Niao He, Yunpeng Pan, Byron Boots, and Le Song (2017). "Learning from Conditional Distributions via Dual Embeddings." In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. PMLR.

Dayan, Peter (1993). "Improving Generalization for Temporal Difference Learning: The Successor Representation." In: *Neural Computation*.

De Asis, Kristopher, Brendan Bennett, and Richard Sutton (2018). *Predicting Periodicity with Temporal Difference Learning*. arXiv: `1809.07435 [cs, eess]`.

Dohare, Shibhansh, Richard Sutton, and A. Rupam Mahmood (2022). *Continual Backprop: Stochastic Gradient Descent with Persistent Randomness*. arXiv: `2108.06325 [cs]`.

Downey, Carlton, Ahmed Hefny, Byron Boots, Geoffrey J Gordon, and Boyue Li (2017). "Predictive State Recurrent Neural Networks." In: *Advances in Neural Information Processing Systems 30*.

Drescher, Gary L. (1991). *Made-Up Minds : A Constructivist Approach to Artificial Intelligence*. Artificial Intelligence. Cambridge, Mass: The MIT Press.

Durkin, John (1996). "Expert Systems: A View of the Field." In: *IEEE Intelligent Systems*.

Durrant-Whyte, H. and T. Bailey (2006). "Simultaneous Localization and Mapping: Part I." In: *IEEE Robotics & Automation Magazine*.

Edwards, Ann L, Michael R Dawson, Jacqueline S Hebert, Craig Sherstan, Richard Sutton, K Ming Chan, and Patrick M Pilarski (2016). "Application of Real-Time Machine Learning to Myoelectric Prosthesis Control: A Case Series in Adaptive Switching." In: *Prosthetics and Orthotics International*.

Espeholt, Lasse, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu (2018). "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures." In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR.

Fedus, William, Carles Gelada, Yoshua Bengio, Marc G. Bellemare, and Hugo Larochelle (2019). *Hyperbolic Discounting and Learning over Multiple Horizons*. arXiv: `1902.06865 [cs, stat]`.

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks." In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR.

Ghiassian, Sina, Andrew Patterson, Martha White, Richard Sutton, and Adam White (2018). *Online Off-policy Prediction*. arXiv: `1811.02597 [cs, stat]`.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the Difficulty of Training Deep Feedforward Neural Networks." In: *Proceedings of the Thir-*

teenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings.

Gordon, N. J., D. J. Salmond, and A. F. M. Smith (1993). “Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation.” In: *IEE Proceedings F (Radar and Signal Processing)*.

Goudreau, M.W., C.L. Giles, S.T. Chakradhar, and D. Chen (1994). “First-Order versus Second-Order Single-Layer Recurrent Neural Networks.” In: *IEEE Transactions on Neural Networks*.

Greff, Klaus, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber (2017). “LSTM: A Search Space Odyssey.” In: *IEEE Transactions on Neural Networks and Learning Systems*.

Günther, Johannes, Patrick M. Pilarski, Gerhard Helfrich, Hao Shen, and Klaus Diepold (2016). “Intelligent Laser Welding through Representation, Prediction, and Control Learning: An Architecture with Deep Neural Networks and Reinforcement Learning.” In: *Mechatronics*. System-Integrated Intelligence: New Challenges for Product and Production Engineering.

Gupta, Dhawal, Gabor Mihucz, Matthew Schlegel, James Kostas, Philip S Thomas, and Martha White (2021). “Structural Credit Assignment in Neural Networks using Reinforcement Learning.” In: *Advances in Neural Information Processing Systems* 34.

Hallak, Assaf and Shie Mannor (2017). “Consistent On-Line Off-Policy Evaluation.” In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR.

Hausknecht, Matthew and Peter Stone (2015). “Deep Recurrent Q-Learning for Partially Observable MDPs.” In: *2015 AAAI Fall Symposium Series*.

Hawkins, Jeff and Sandra Blakeslee (2004). *On Intelligence*. Macmillan.

Heess, Nicolas, Jonathan J. Hunt, Timothy P. Lillicrap, and David Silver (2015). “Memory-Based Control with Recurrent Neural Networks.” In: *NeurIPS Deep Reinforcement Learning Workshop*. arXiv: `1512.04455`.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-Term Memory.” In: *Neural Computation*.

Hopfield, J J (1982). “Neural Networks and Physical Systems with Emergent Collective Computational Abilities.” In: *Proceedings of the National Academy of Sciences*.

Hsu, Daniel, Sham M. Kakade, and Tong Zhang (2012). “A Spectral Algorithm for Learning Hidden Markov Models.” In: *Journal of Computer and System Sciences*. JCSS Special Issue: Cloud Computing 2011.

Huang, Yanping and Rajesh P. N. Rao (2011). “Predictive Coding.” In: *Wiley Interdisciplinary Reviews: Cognitive Science*.

Igl, Maximilian, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson (2018). “Deep Variational Reinforcement Learning for POMDPs.” In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR.

Innes, Mike (2018). “Flux: Elegant Machine Learning with Julia.” In: *Journal of Open Source Software*.

Jacobsen, Andrew, Matthew Schlegel, Cameron Linke, Thomas Degris, Adam White, and Martha White (2019). "Meta-descent for online, continual prediction." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01, pp. 3943–3950.

Jaderberg, Max, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu (2017). "Reinforcement Learning with Unsupervised Auxiliary Tasks." In: *International Conference on Representation Learning*.

Jaeger, Herbert and Harald Haas (2004). "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication." In: *Science*.

James, Michael, Britton Wolfe, and Satinder Singh (2005). "Combining Memory and Landmarks with Predictive State Representations." In: *IJCAI*.

Javed, Khurram (2020). "Learning Online-Aware Representations Using Neural Networks." PhD thesis. University of Alberta.

Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra (1998). "Planning and Acting in Partially Observable Stochastic Domains." In: *Artificial Intelligence*.

Kahn, H. and A. W. Marshall (1953). "Methods of Reducing Sample Size in Monte Carlo Computations." In: *Journal of the Operations Research Society of America*.

Kapturowski, Steven, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney (2019). "Recurrent Experience Replay in Distributed Reinforcement Learning." In: *International Conference on Learning Representations*.

Karpathy, Andrej, Justin Johnson, and Li Fei-Fei (2015). *Visualizing and Understanding Recurrent Networks*. arXiv: `1506.02078 [cs]`.

Ke, Nan Rosemary, Anirudh Goyal ALIAS PARTH GOYAL, Olexa Bilaniuk, Jonathan Binas, Michael C Mozer, Chris Pal, and Yoshua Bengio (2018). "Sparse Attentive Backtracking: Temporal Credit Assignment Through Reminding." In: *Advances in Neural Information Processing Systems 31*.

Kearney, Alex and Oliver Oxton (2019). *Making Meaning: Semiotics Within Predictive Knowledge Architectures*. arXiv: `1904.09023 [cs]`.

Khetarpal, Khimya, Matthew Riemer, Irina Rish, and Doina Precup (2022). "Towards Continual Reinforcement Learning: A Review and Perspectives." In: *Journal of Artificial Intelligence Research*.

Kong, Augustine, Jun S. Liu, and Wing Hung Wong (1994). "Sequential Imputations and Bayesian Missing Data Problems." In: *Journal of the American Statistical Association*.

Kudenko, Daniel and Haym Hirsh (1998). "Feature Generation for Sequence Categorization." PhD thesis. USA: Rutgers University.

Kumaraswamy, Raksha, Matthew Schlegel, Adam White, and Martha White (2018). "Context-dependent upper-confidence bounds for directed exploration." In: *Advances in Neural Information Processing Systems* 32.

LeCun, Yann, Corinna Cortes, and CJ Burges (2010). *MNIST Handwritten Digit Database*.

Lin, Long-Ji (1992). “Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching.” In: *Machine Learning.*

— (1993). “Reinforcement Learning for Robots Using Neural Networks.” PhD thesis. Carnegie Mellon University.

Linke, Cam, Nadia M. Ady, Martha White, Thomas Degris, and Adam White (2020). “Adapting Behavior via Intrinsic Reward: A Survey and Empirical Study.” In: *Journal of Artificial Intelligence Research.*

Littman, Michael L. and Richard Sutton (2002). “Predictive Representations of State.” In: *Advances in Neural Information Processing Systems 14.* MIT Press.

Liu, Qiang, Lihong Li, Ziyang Tang, and Dengyong Zhou (2018). “Breaking the Curse of Horizon: Infinite-Horizon Off-Policy Estimation.” In: *Advances in Neural Information Processing Systems 31.*

López, Victoria, Alberto Fernández, Salvador García, Vasile Palade, and Francisco Herrera (2013). “An Insight into Classification with Imbalanced Data: Empirical Results and Current Trends on Using Data Intrinsic Characteristics.” In: *Information Sciences.*

Maaten, Laurens van der and Geoffrey Hinton (2008). “Visualizing Data Using T-SNE.” In: *Journal of Machine Learning Research.*

Maei, Hamid, Csaba Szepesvári, Shalabh Bhatnagar, David Silver, and Richard Sutton (2009). “Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation.” In: *Advances in Neural Information Processing Systems.*

Maei, Hamid, Csaba Szepesvári, Shalabh Bhatnagar, and Richard Sutton (2010). “Toward Off-policy Learning Control with Function Approximation.” In: *Proceedings of the 27th International Conference on Machine Learning.* PMLR.

Mahmood, A. Rupam and Richard Sutton (2013). “Representation Search through Generate and Test.” In: *AAAI Workshop: Learning Rich Representations from Low-Level Sensors.*

— (2015). “Off-Policy Learning Based on Weighted Importance Sampling with Linear Computational Complexity.” In: *UAI.*

Mahmood, A. Rupam, Hado van Hasselt, and Richard Sutton (2014). “Weighted Importance Sampling for Off-Policy Learning with Linear Function Approximation.” In: *Advances in Neural Information Processing Systems 27.*

Mahmood, A. Rupam, Huizhen Yu, and Richard Sutton (2017). “Multi-Step Off-policy Learning Without Importance Sampling Ratios.” In: *arXiv:1702.03006 [cs].* arXiv: `1702.03006 [cs]`.

Makino, Takaki and Toshihisa Takagi (2008). “On-Line Discovery of Temporal-difference Networks.” In: *Proceedings of the 25th International Conference on Machine Learning.* PMLR.

Martino, L., V. Elvira, and F. Louzada (2017). “Effective Sample Size for Importance Sampling Based on Discrepancy Measures.” In: *Signal Processing.*

McCallum, Andrew (1993). “Overcoming Incomplete Perception with Util Distinction Memory.” In: *Proceedings of the 10th International Conference on Machine Learning*. PMLR.

McCallum, Andrew et al. (1996). “Learning to Use Selective Attention and Short-Term Memory in Sequential Tasks.” In: *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*.

Mccracken, Peter and Michael Bowling (2006). “Online Discovery and Learning of Predictive State Representations.” In: *Advances in Neural Information Processing Systems 18*. MIT Press.

McLeod, Matthew, Chunlok Lo, Matthew Schlegel, Andrew Jacobsen, Raksha Kumaraswamy, Martha White, and Adam White (2021). “Continual Auxiliary Task Learning.” In: *Advances in Neural Information Processing Systems* 34.

Menick, Jacob, Erich Elsen, Utku Evci, Simon Osindero, Karen Simonyan, and Alex Graves (2020). *A Practical Sparse Approximation for Real Time Recurrent Learning*. arXiv: `2006.07232 [cs, stat]`.

Mikolov, Tomas, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur (2010). “Recurrent Neural Network Based Language Model.” In: *Interspeech*. Makuhari.

Ming, Yao, Shaozu Cao, Ruixiang Zhang, Zhen Li, Yuanzhe Chen, Yangqiu Song, and Huamin Qu (2017). “Understanding Hidden Memories of Recurrent Neural Networks.” In: *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*.

Mishra, Nikhil, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel (2018). “A Simple Neural Attentive Meta-Learner.” In: *International Conference on Learning Representations*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis (2015). “Human-Level Control through Deep Reinforcement Learning.” In: *Nature*.

Modayil, Joseph, Adam White, and Richard Sutton (2014). “Multi-Timescale Nexting in a Reinforcement Learning Robot.” In: *Adaptive Behavior*.

Momennejad, Ida and Marc W. Howard (2018). *Predicting the Future with Multi-scale Successor Representations*.

Mozer, Michael C (1991). “Induction of Multiscale Temporal Structure.” In: *Advances in Neural Information Processing Systems 4*.

— (1995). “A Focused Backpropagation Algorithm for Temporal.” In: *Backpropagation: Theory, architectures, and applications*.

Mujika, Asier, Florian Meier, and Angelika Steger (2018). “Approximating Real-Time Recurrent Learning with Random Kronecker Factors.” In: *Advances in Neural Information Processing Systems*.

Munos, Remi, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare (2016). "Safe and Efficient Off-Policy Reinforcement Learning." In: *Advances in Neural Information Processing Systems 29*.

Nath, Somjit, Vincent Liu, Alan Chan, Xin Li, Adam White, and Martha White (2020). "Training Recurrent Neural Networks Online by Learning Explicit State Variables." In: *International Conference on Learning Representations*.

Oh, Junhyuk, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh (2015). "Action-Conditional Video Prediction Using Deep Networks in Atari Games." In: *Advances in Neural Information Processing Systems 28*.

Onat, A., H. Kita, and Y. Nishikawa (1998). "Recurrent Neural Networks for Reinforcement Learning: Architecture, Learning Algorithms and Internal Representation." In: *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*.

Oppenheim, Alan V and Ronald W Schafer (2010). *Discrete-Time Signal Processing*. Pearson Higher Education.

Owen, Art B. (2013). *Monte Carlo Theory, Methods and Examples*. Chap. 2.

Parisotto, Emilio, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphaël Lopez Kaufman, Aidan Clark, Seb Noury, Matthew Botvinick, Nicolas Heess, and Raia Hadsell (2020). "Stabilizing Transformers for Reinforcement Learning." In: *Proceedings of the 37th International Conference on Machine Learning*. PMLR.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). "On the Difficulty of Training Recurrent Neural Networks." In: *Proceedings of the 30th International Conference on Machine Learning*. PMLR.

Patterson, Andrew, Adam White, and Martha White (2022). "A Generalized Projected Bellman Error for Off-policy Value Estimation in Reinforcement Learning." In: *Journal of Machine Learning Research*.

Pearlmutter, Barak A. (1994). "Fast Exact Multiplication by the Hessian." In: *Neural Computation*.

Peng, J. and R.J. Williams (1993). "Efficient Learning and Planning within the Dyna Framework." In: *IEEE International Conference on Neural Networks*.

Pezzulo, Giovanni (2008). "Coordinating with the Future: The Anticipatory Nature of Representation." In: *Minds and Machines*.

Pezzulo, Giovanni, Martin V. Butz, Cristiano Castelfranchi, and Rino Falcone (2008). *The Challenge of Anticipation: A Unifying Framework for the Analysis and Design of Artificial Cognitive Systems*. Springer.

Precup, Doina, Richard Sutton, and Sanjoy Dasgupta (2001). "Off-Policy Temporal-Difference Learning with Function Approximation." In: *Proceedings of the 18th International Conference on Machine Learning*. PMLR.

Precup, Doina, Richard Sutton, and Satinder Singh (1998). "Theoretical Results on Reinforcement Learning with Temporally Abstract Options." In: *Machine Learning: ECML-98*. 10th European Conference on Machine Learning. Berlin, Heidelberg: Springer.

Precup, Doina, Richard Sutton, and Satinder Singh (2000). "Eligibility Traces for Off-Policy Policy Evaluation." In: *Computer Science Department Faculty Publication Series*.

Rafiee, Banafsheh, Zaheer Abbas, Sina Ghiassian, Raksha Kumaraswamy, Richard Sutton, Elliot A Ludvig, and Adam White (2022). "From Eye-Blinks to State Construction: Diagnostic Benchmarks for Online Representation Learning." In: *Adaptive Behavior*.

Rafiee, Banafsheh, Sina Ghiassian, Jun Jin, Richard Sutton, Jun Luo, and Adam White (2022). *Auxiliary Task Discovery through Generate-and-Test*. arXiv: `2210.14361 [cs]`.

Rafols, Eddie, Anna Koop, and Richard Sutton (2006). "Temporal Abstraction in Temporal-difference Networks." In: *Advances in Neural Information Processing Systems 18*.

Rafols, Eddie, Mark Ring, Richard Sutton, and Brian Tanner (2005). "Using Predictive Representations to Improve Generalization in Reinforcement Learning." In: *IJCAI*.

Rao, Rajesh P. N. and Dana H. Ballard (1999). "Predictive Coding in the Visual Cortex: A Functional Interpretation of Some Extra-Classical Receptive-Field Effects." In: *Nature Neuroscience*.

Riedmiller, Martin, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Wiele, Vlad Mnih, Nicolas Heess, and Jost Tobias Springenberg (2018). "Learning by Playing Solving Sparse Reward Tasks from Scratch." In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR.

Ring, Mark (1994). "Continual Learning In Reinforcement Environments." PhD thesis. University of Texas at Austin.

— (1997). "CHILD: A First Step Towards Continual Learning." In: *Machine Learning*.

Rubin, Donald B (1988). "Using the SIR Algorithm to Simulate Posterior Distribution." In: *Bayesian statistics*.

Rubinstein, Reuven Y and Dirk P Kroese (2016). *Simulation and the Monte Carlo Method*. John Wiley & Sons.

Rummery, Gavin A and Mahesan Niranjan (1994). *On-Line Q-learning Using Connectionist Systems*. University of Cambridge, Department of Engineering Cambridge, UK.

Russek, Evan M., Ida Momennejad, Matthew M. Botvinick, Samuel J. Gershman, and Nathaniel D. Daw (2017). "Predictive Representations Can Link Model-Based Reinforcement Learning to Model-Free Mechanisms." In: *PLOS Computational Biology*.

Samani, Amir and Richard Sutton (2021). *Learning Agent State Online with Recurrent Generate-and-Test*. arXiv: `2112.15236 [cs]`.

Saon, George, Gakuto Kurata, Tom Sercu, Kartik Audhkhasi, Samuel Thomas, Dimitrios Dimitriadis, Xiaodong Cui, Bhuvana Ramabhadran, Michael Picheny, Lynn-Li Lim, Bergul Roomi, and Phil Hall (2017). "English Conversational Telephone Speech Recognition by Humans and Machines." In: *Proc. Interspeech 2017*.

Schaefer, Anton Maximilian, Steffen Udluft, and Hans-Georg Zimmermann (2007). "A Recurrent Control Neural Network for Data Efficient Reinforcement Learning." In: *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*.

Schaul, Tom, Daniel Horgan, Karol Gregor, and David Silver (2015). "Universal Value Function Approximators." In: *Proceedings of the 32nd International Conference on Machine Learning*. PMLR.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver (2015). "Prioritized Experience Replay." In: *arXiv:1511.05952 [cs]*. arXiv: `1511.05952 [cs]`.

Schaul, Tom and Mark Ring (2013). "Better Generalization with Forecasts." In: *IJCAI*.

Schlegel, Matthew, Wesley Chung, Daniel Graves, Jian Qian, and Martha White (2019). "Importance resampling for off-policy prediction." In: *Advances in Neural Information Processing Systems* 32.

Schlegel, Matthew, Andrew Jacobsen, Zaheer Abbas, Andrew Patterson, Adam White, and Martha White (2021). "General value function networks." In: *Journal of Artificial Intelligence Research*.

Schlegel, Matthew, Volodymyr Tkachuk, Adam M White, and Martha White (2022). "Investigating Action Encodings in Recurrent Neural Networks in Reinforcement Learning." In: *Transactions on Machine Learning Research*.

Schlegel, Matthew and Martha White (2022). "Predictions Predicting Predictions." In: *Multidisciplinary Conference on Reinforcement Learning and Decision Making*.

Schütz-Bosbach, Simone and Wolfgang Prinz (2007). "Prospective Coding in Event Representation." In: *Cognitive Processing*.

Sherstan, Craig, Marlos C. Machado, and Patrick M. Pilarski (2018). "Accelerating Learning in Constructive Predictive Frameworks with the Successor Representation." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Silver, David (2013). "Gradient Temporal Difference Networks." In: *European Workshop on Reinforcement Learning*.

Silver, David, Hado Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, and Thomas Degris (2017). "The Predictron: End-To-End Learning and Planning." In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR.

Singh, Satinder, Michael James, and Matthew R. Rudary (2004). "Predictive State Representations: A New Theory for Modeling Dynamical Systems." In: *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*.

Singh, Satinder and Richard Sutton (1996). "Reinforcement Learning with Replacing Eligibility Traces." In: *Machine Learning*.

Skare, Øivind, Erik Bølviken, and Lars Holden (2003). “Improved Sampling-Importance Resampling and Reduced Bias Importance Sampling.” In: *Scandinavian Journal of Statistics.*

Smith, A. F. M. and A. E. Gelfand (1992). “Bayesian Statistics without Tears: A Sampling-Resampling Perspective.” In: *The American Statistician.* JSTOR: 2684170.

Soga, Ryosuke, Rei Akaishi, and Katsuyuki Sakai (2009). “Predictive and Postdictive Mechanisms Jointly Contribute to Visual Awareness.” In: *Consciousness and Cognition.*

Stolle, Martin and Doina Precup (2002). “Learning Options in Reinforcement Learning.” In: *International Symposium on Abstraction, Reformulation, and Approximation.* Springer.

Subramanian, Jayakumar, Amit Sinha, Raihan Seraj, and Aditya Mahajan (2022). “Approximate Information State for Approximate Planning and Reinforcement Learning in Partially Observed Systems.” In: *Journal of Machine Learning Research.*

Sun, Wen, Arun Venkatraman, Byron Boots, and J. Andrew Bagnell (2016). “Learning to Filter with Predictive State Inference Machines.” In: *Proceedings of The 33rd International Conference on Machine Learning.* PMLR.

Sutskever, Ilya, James Martens, and Geoffrey Hinton (2011). “Generating Text with Recurrent Neural Networks.” In: *Proceedings of the 28th International Conference on Machine Learning.* PMLR.

Sutton, Richard (1995). “TD Models: Modeling the World at a Mixture of Time Scales.” In: *Machine Learning Proceedings.*

Sutton, Richard and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction.* Second edition. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press.

Sutton, Richard, Hamid Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora (2009). “Fast Gradient-Descent Methods for Temporal-Difference Learning with Linear Function Approximation.” In: *Proceedings of the 26th Annual International Conference on Machine Learning.* ACM. PMLR.

Sutton, Richard, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup (2011). “Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction.” In: *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2.* International Foundation for Autonomous Agents and Multiagent Systems.

Sutton, Richard, Doina Precup, and Satinder Singh (1999). “Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning.” In: *Artificial Intelligence.*

Sutton, Richard and Brian Tanner (2005). “Temporal-Difference Networks.” In: *Advances in Neural Information Processing Systems 17.*

Synofzik, Matthis, Gottfried Vosgerau, and Martin Voss (2013). "The Experience of Agency: An Interplay between Prediction and Postdiction." In: *Frontiers in Psychology*.

Tallec, Corentin and Yann Ollivier (2018). "Unbiased Online Recurrent Optimizatin." In: *International Conference on Learning Representations*.

Tanner, Brian (2005). "Temporal-Difference Networks." PhD thesis. University of Alberta.

Tanner, Brian and Richard Sutton (2005). "Temporal-Difference Networks with History." In: *IJCAI*.

Thomas, Philip and Emma Brunskill (2016). "Data-Efficient Off-Policy Policy Evaluation for Reinforcement Learning." In: *Proceedings of The 33rd International Conference on Machine Learning*. PMLR.

— (2017). "Importance Sampling with Unequal Support." In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Trinh, Trieu, Andrew Dai, Thang Luong, and Quoc Le (2018). "Learning Longer-term Dependencies in RNNs with Auxiliary Losses." In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR. Chap. Machine Learning.

van Hasselt, Hado, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil (2018). "Deep Reinforcement Learning and the Deadly Triad." In: *arXiv:1812.02648 [cs]*. arXiv: `1812.02648 [cs]`.

van Hasselt, Hado and Richard Sutton (2015). "Learning to Predict Independent of Span." In: *arXiv:1508.04582 [cs]*. arXiv: `1508.04582 [cs]`.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention Is All You Need." In: *Advances in Neural Information Processing Systems 31*.

Veeriah, Vivek, Matteo Hessel, Zhongwen Xu, Janarthanan Rajendran, Richard L Lewis, Junhyuk Oh, Hado van Hasselt, David Silver, and Satinder Singh (2019). "Discovery of Useful Questions as Auxiliary Tasks." In: *Advances in Neural Information Processing Systems 32*.

Venkatraman, Arun, Nicholas Rhinehart, Wen Sun, Lerrel Pinto, Martial Hebert, Byron Boots, Kris Kitani, and J. Bagnell (2017). "Predictive-State Decoders: Encoding the Future into Recurrent Networks." In: *Advances in Neural Information Processing Systems 30*.

Wang, Tian and Kyunghyun Cho (2016). "Larger-Context Language Modelling with Recurrent Neural Network." In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Wang, Yunbo, Lu Jiang, Ming-Hsuan Yang, Li-Jia Li, Mingsheng Long, and Li Fei-Fei (2018). "Eidetic 3D LSTM: A Model for Video Prediction and Beyond." In: *International Conference on Learning Representations*.

Watkins, Christopher J. C. H. and Peter Dayan (1992). "Q-Learning." In: *Machine Learning*.

White, Adam (2015). “Developing a Predictive Approach to Knowledge.” PhD thesis. University of Alberta.

White, Martha (2017). “Unifying Task Specification in Reinforcement Learning.” In: *Proceedings of the 34th International Conference on Machine Learning.* PMLR.

Wierstra, Daan, Alexander Foerster, Jan Peters, and Jürgen Schmidhuber (2007). “Solving Deep Memory POMDPs with Recurrent Policy Gradients.” In: *Artificial Neural Networks – ICANN 2007.* Lecture Notes in Computer Science. Berlin, Heidelberg: Springer.

Williams, Ronald J. and David Zipser (1989). “A Learning Algorithm for Continually Running Fully Recurrent Neural Networks.” In: *Neural Computation.*

Wingate, David and Satinder Singh (2007). “On Discovery and Learning of Models with Predictive Representations of State for Agents with Continuous Actions and Observations.” In: *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems.* ACM.

Wolfe, Britton, Michael James, and Satinder Singh (2005). “Learning Predictive State Representations in Dynamical Systems Without Reset.” In: *Proceedings of the 22nd International Conference on Machine Learning.* PMLR.

Wolfe, Britton and Satinder Singh (2006). “Predictive State Representations with Options.” In: *Proceedings of the 23rd International Conference on Machine Learning.* PMLR.

Zhu, Pengfei, Xin Li, Pascal Poupart, and Guanghui Miao (2018). “On Improving Deep Reinforcement Learning for POMDPs.” In: *arXiv:1704.07978 [cs].* arXiv: 1704.07978 [cs].

Appendix A

Details and Extended Ideas for Incorporating Actions in Recurrent Networks

A.1 Online Setting

In this section, we test to see if our conclusions from the previous sections generalize to the fully online setting. For both environments, all applicable settings are the same as in the replay counter parts. The only difference is in how the network is updated. Instead of sampling from an experience replay, we store a history of the truncation length and update the network on every step using the same semi-gradient updates.

We present the results for the online setting in figure A.1. Compared to the replay setting, we can see all the variants performed worse across the board. For DirectionalTMaze the AAGRU and MAGRU have a reasonable median performance. The MARNN and FacGRU are the only other cells which have runs reaching good performance, but overall perform poorly. We expect initialization plays a large role in the networks performance and should be investigated. We also see similar trends in Ring World, except the RNN variants outperform the GRUs. Another interesting consequence in the online setting, is the need to increase the truncation value and hidden state size to perform reasonably for both DirectionalTMaze and Ring World.

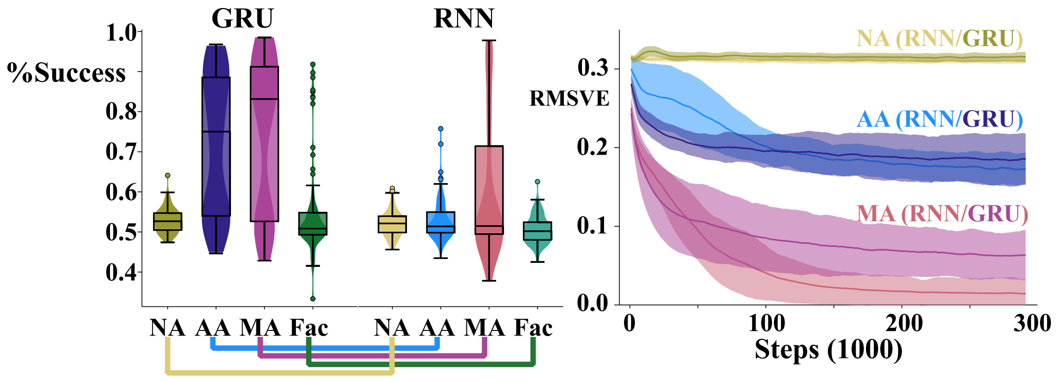


Figure A.1: Online: **(left + middle)** Directional TMaze percent success in reaching the goal over the final 10% of episodes with 100 independent runs for CELL (hidden size): RNN (46), AARNN (46), MARNN (27), FacRNN (46) $M = 24$, GRU (26), AAGRU (26), MAGRU (15), FacGRU (26) $M = 21$. **(right)** Ring World learning curves over RMSVE with 100 independent runs for: RNN (20), AARNN (20), MARNN (15), GRU (12), AAGRU (12), MAGRU (9). Ribbons show standard error and a window averaging over 10k steps was used. Factored variants were excluded for clarity, due to high variance results. All agents were trained over 300k steps.

A.2 Masked Grid World

While the TMaze and DirTMaze give some insight into when different encodings might be preferable, the DirTMaze and Ring World share similar dynamics in how the actions effect the unobserved state of the MDP. Specifically, there are two actions which effect a state component symmetrically. This prompts the question on whether this property is driving the benefits of the multiplicative update's success, or whether there are other scenarios where the multiplicative does better. We propose a new environment which is a simple grid world with border wrapping. The agent can take a step in all the cardinal directions, and observes when it enters a random subset of the states (all aliased together). The goal state is also randomly selected at the beginning of an agent's life. This creates random action observation patterns the agent must notice and act on to get to the goal. The border wrapping prevents the agent from moving to a corner of the environment and then going to the goal.

In figure A.2, we confirm the hypothesis that the improvement with multiplicative update can be meaningful even when the state-action sequences are

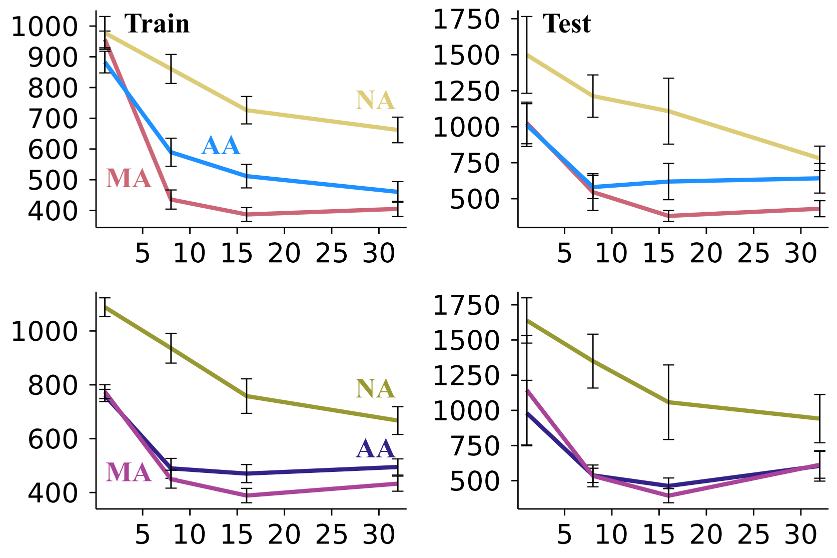


Figure A.2: Average number of steps to goal over truncation for Masked Grid World **left** over the entire learning process **right** from a set of representative states after training. Cell (state size) **top** RNN (24), AARNN (20), MARNN (10) **bottom** GRU (24), AAGRU (20), MAGRU (10).

randomly placed in the environment. While the improvement is much less drastic than the Ring World and DirTMaze, the improvement is still significant with standard error bars. Another interesting observation is the difference matters much more for the simple recurrent update than the GRU.

A.3 Further Results for the Deep Additive and Deep Multiplicative Architectures

We provide two more experiments with results reported in figure A.3. First we provide results over various action encoding sizes for the Directional TMaze environment using the Deep Additive network from the main paper. Overall, we found the size of the encoding network to not make a large difference in the final performance. In effect, this result suggests there is still a core limitation with the deep additive operation that can't be overcome by larger encoding networks. We also provide an experiment in the Directional TMaze for a **deep multiplicative** update. The deep multiplicative update uses the multiplicative update as a base cell but first passes the action through a feed forward network

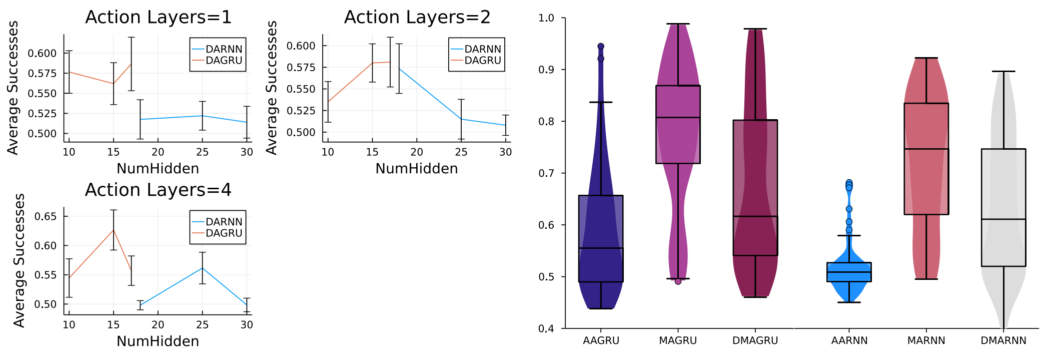


Figure A.3: **(left)** Resulting average success over the last 10% of episodes for various number of hidden units in the action encoding network for the deep additive networks with standard error intervals. Each layer (denoted by the title of the plot) contains the number of hidden denoted by the x-axis. **(right)** Comparing the deep multiplicative operation with the base cells used in the main text.

like the deep additive network. The results of this network were quite poor overall, likely as a result of having to learn the action encoding rather than being given it as prior information. From these results we decided to abandon the deep multiplicative extension of Zhu et al. 2018. Future work should consider how to better learn action encodings for such a network.

A.4 Further Empirical Details

In this section we discuss the experiments from the paper in greater detail. In the following tables the common programming notation $(x : y : z)$ is used to denote an array of elements starting from x increasing by y until z. For example $(1 : 2 : 5) = [1, 3, 5]$. When an operation is performed on an array it is done element wise. For example $2^{(1:2:5)} = [2, 4, 32]$. All hyperparameters are reported in agent steps (which are the same as environment steps for all domains).

A.4.1 Ring World

Table A.1 gives the hyperparameters used in the ringworld experiments. We also provide full sensitivity curves over truncation for all cell types and hidden state sizes tested in Figure A.4.

Parameter	Value
Steps	300,000 steps
Optimizer	RMSprop
RMSProp η	$0.1 \times 1.6^{(-16:3:-2)}$
RMSprop ρ	0.9
Buffer Size	1000
Buffer Warmup	1000
Batch Size	4
Update freq	4 steps
Target Network Freq	1000 steps
Independent Runs	50

Table A.1: Ring World Hyperparameters

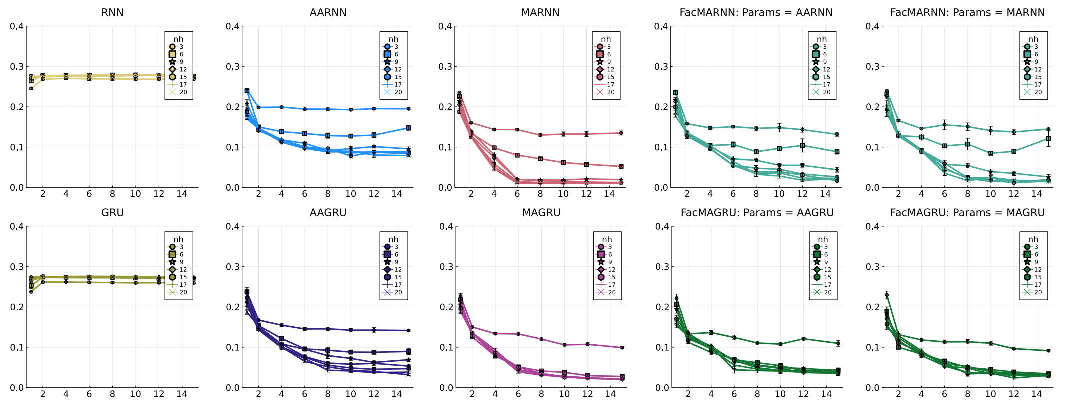


Figure A.4: Truncation sensitivity curves for the Experience Replay setting in ring world. Results are RMSVE and error bars are 95% confidence, as in the main paper.)

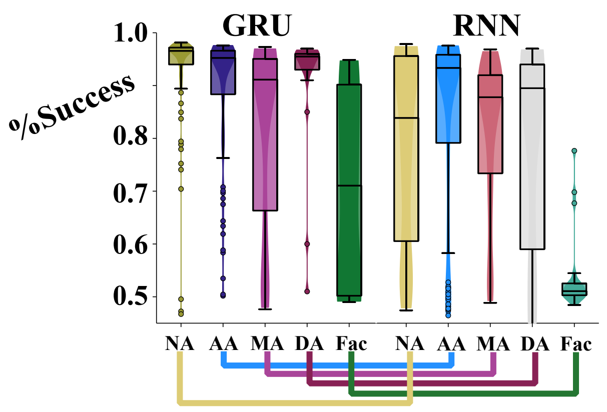


Figure A.5: Extended TMaze Experience Replay experiments.

A.4.2 TMaze

In Figure A.2 we provide details of the experience replay of the Bakker's TMaze experiments.

A.4.3 DirectionalTMaze

The experiments presented in the paper used an experience replay buffer size of 10000 for all the cells. We also ran experiments using an experience replay size of 20000 with similar conclusions. These results (and the associated parameters) can be found in Figure A.6

A.4.4 Image Directional TMaze

We detail all hyperparameter settings, and give results for different network sizes and truncation values in Figure A.7.

A.4.5 Lunar Lander

We provide all hyperparameter settings (Figure A.8) and further results (Figure A.9).

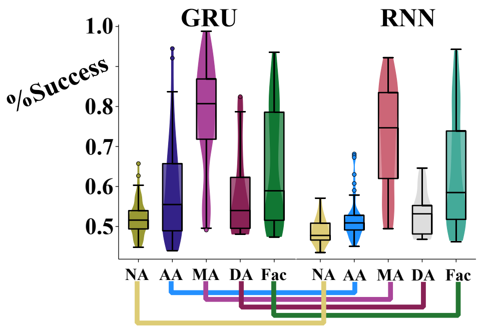
Parameter	Value
Steps	300,000 steps
Optimizer	RMSprop
RMSProp RNN: η	$0.01 \times (2.0^{(-11:2:-2)})$
RMSProp GRU: η	$0.01 \times (2.0^{(-11:2:-6)})$
RMSprop ρ	0.99
Discount γ	0.99
Truncation τ	12
Buffer Size	10000
Batch Size	8
Update freq	4 steps
Target Network Freq	1000 steps
Independent Runs	50

(a) Shared Hyperparameters.

Cell	RMSprop Learning Rate	Hidden State Size	Model Parameters
GRU	0.005	6	214
AAGRU	0.0003125	6	286
MAGRU	0.0003125	6	754
FacGRU	0.0003125	6, $M = 21$	757
DAGRU	0.00125	6, $a = 4$	306
RNN	7.8125e-5	20	584
AARNN	7.8125e-5	20	664
MARNN	7.8125e-5	20	2024
FacRNN	0.0003125	20, $M = 40$	2064
DARNN	7.8125e-5	20, $a = 4$	684

(b) Architecture specific hyperparameters.

Table A.2: Hyperparameters used in the TMaze Experience Replay experiments.



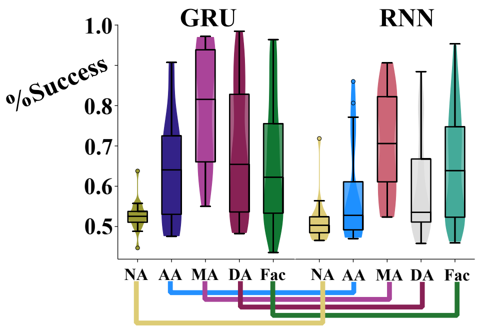


Figure A.6: Directional TMaze Experience Replay results: **(left)** Percent success over the final 10% of episodes for buffer size of 10000 **(right)** for buffer size of 20000. Learning rates chosen from best final performance on the final 10% of episodes for 20 runs. Results for buffer size of 10000 are over 100 independent runs, with buffer size of 20000 in appendix only over the 20 seeds used for the sweep.

Parameter	Value
Steps	300,000 steps
Optimizer	RMSprop
RMSprop η	$0.01 \times 2.0^{(-11:2:-2)}$
RMSprop ρ	0.99
Discount γ	0.99
Truncation τ	12
Buffer Size	[10000, 20000]
Batch Size	8
Update freq	4 steps
Target update freq	1000 steps
Independent Runs	100

(a) Shared hyperparameteters.

Cell	η	Hidden State Size	Num Weights
GRU	0.00125	17	1142
AAGRU	0.0003125	17	1295
MAGRU	0.0003125	10	1303
FacGRU	0.0003125	15, $M = 17$	1320
DAGRU	0.0003125	15, $a = 8$	1310
RNN	0.00125	30	1143
AARNN	0.0003125	30	1233
MARNN	7.8125e-5	18	1263
FacRNN	0.0003125	25, $M = 15$	1018
DARNN	0.0003125	25, $a = 15$	1263

(b) Architecture specific hyperparameters.

Table A.3: Hyperparameters used in the Direction TMaze experience replay experiments.

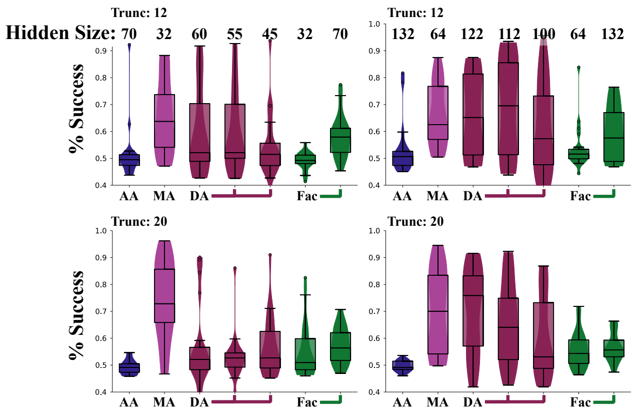


Figure A.7: Image Directional TMaze: Percent success over final 10% of episodes for the image tmaze for $\tau = 12$ and $\tau = 20$ (labeled). See labels for size of hidden state with left being small networks, and right being larger.

Parameter	Value
Steps	400,000
Optimizer	RMSprop
ADAM η	$2.0^{(-20:2:0)}$
ADAM β	0.9, 0.999
Discount γ	0.99
Truncation τ	12 (20)
Warm up	1000 steps
Replay Size	50,000
Batch size	16
Target update freq	10000
Update freq	4
Runs	20

(a) Shared hyperparameteters.

Cell	η $\tau = 12(20)$	$\lvert s\rvert$ + M	$\lvert\mathbf{W}\rvert$
MAGRU	9.76e-4 (9.76e-4)	32	154247
MAGRU	9.76e-4 (9.76e-4)	64	223175
AAGRU	2.44e-4 (0.0625)	70	155201
AAGRU	9.76e-4 (0.0156)	132	225323
FacGRU	0.0625 (2.44e-4)	32, M=259	154224
FacGRU	9.76e-4 (2.44e-4)	64, M=350	223009
FacGRU	9.76e-4 (2.44e-4)	70, M=164	155041
FacGRU	9.76e-4 (2.44e-4)	132, M=208	225515
DAGRU	9.76e-4 (9.76e-4)	45, a=128	150838
DAGRU	9.76e-4 (9.76e-4)	55, a=64	152022
DAGRU	9.76e-4 (9.76e-4)	60, a=32	151399
DAGRU	9.76e-4 (9.76e-4)	100, a=128	224263
DAGRU	9.76e-4 (9.76e-4)	112, a=64	220935
DAGRU	9.76e-4 (9.76e-4)	122, a=32	223195

(b) Architecture specific hyperparameters.

Table A.4: Hyperparameters used in the Image Directional TMaze experiments.

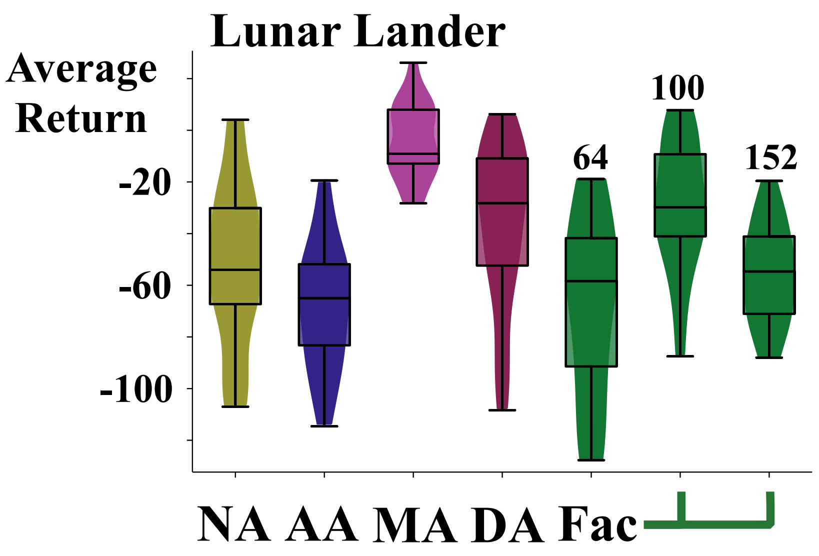


Figure A.8: Lunar lander results including the factored architecture.

Parameter	Value
Steps	4,000,000 steps
Steps before learning	1000 steps
Optimizer	RMSprop
RMSprop η	$0.1 \times 1.6^{(-20:2:-6)}$
RMSprop ρ	0.99
Discount γ	0.99
Truncation τ	16
Replay Size	100,000
Batch size	32
Target update freq	1000 steps
Update frequency	8 steps
Hidden state learnable	True
Independent Runs	20

(a) Shared hyperparameters

Cell	RMSprop Learning Rate	**s** Size $(M/\|a\|)$	Num Weights
GRU	0.0003553	154	156,414
AAGRU	0.0001387	152	155,004
MAGRU	0.0003553	64	153,732
FacGRU	0.0001387	152 (M=170)	152,668
FacGRU	0.0003553	100 (M=265)	153,808
FacGRU	0.0003553	64 (M=380)	153,716
DAAGRU	0.000138778	152 (a=64)	182,684

(b) Architecture specific hyperparameters

Table A.5: Hyperparameters used for Lunar Lander Experiments.

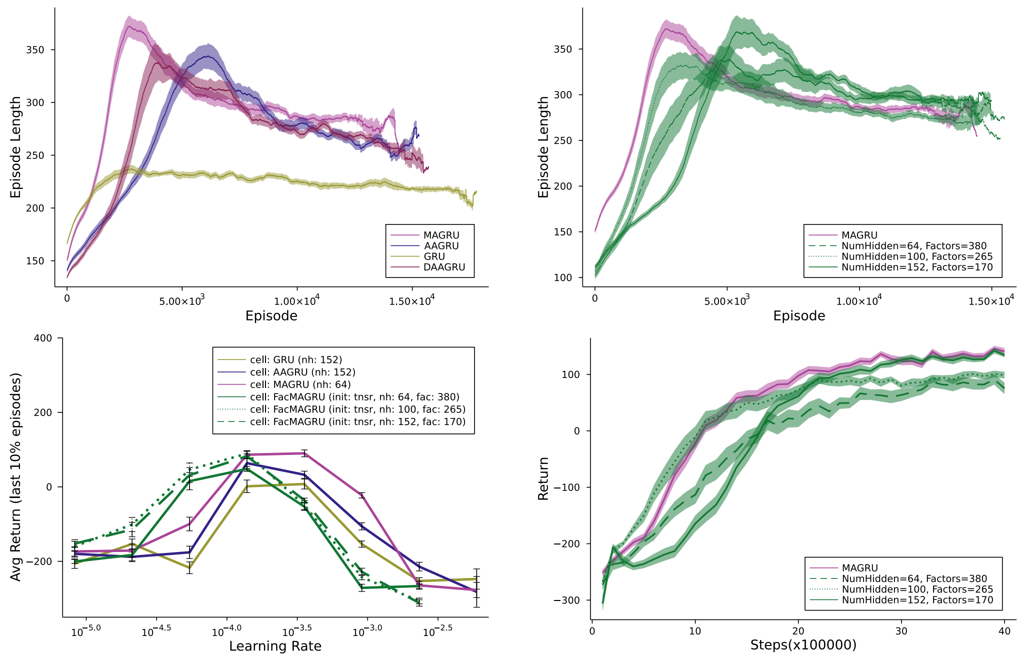


Figure A.9: Lunar Lander further results: **(top left)** Average final reward over the final 10% of episodes for 20 runs **(top middle)** Total steps per episode for non-factored cells for 20 runs **(top right)** Total steps per episode for factored cells for 20 runs **(bottom left)** learning rate sensitivity curves for 10 runs **(bottom middle)** Learning curves per episode for non-factored cells over total reward for 20 runs **(bottom right)** Learning curves per episode for factored cells over total reward for 20 runs

Appendix B

Further Details and Supporting Proofs for Importance Resampling

B.1 Variance of Updates

It might seem that resampling avoids high-variance in updates, because it does not reweight with large magnitude IS ratios. The notion of *effective sample size* from statistics, however, provides some intuition about why large magnitude IS ratios can also negatively affect IR, not just IS. %For data with several high magnitude ratios, and many small ratios, the IS estimator will likely suffer from high-variance updates. IR, however, may not perform well in this setting either: it will prevent high magnitude updates, but will be sampling from an effectively smaller dataset. Effective sample size is between 1 and n, with one estimator $\left(\sum_{i=1}^n \rho_i\right)^2 / \sum_{i=1}^n \rho_i^2$ (Kong et al. 1994; Martino et al. 2017). When the effective sample size is low, this indicates that most of the probability is concentrated on a few samples. For high magnitude ratios, IR will repeatedly sample the same transitions, and potentially never sample some of the transitions with small IS ratios.

Fortunately, we find that, despite this dependence on effective sample size, IR can significantly reduce variance over IS. In this section, we characterize the variance of the BC-IR estimator. We choose this variant of IR, because it is unbiased and so characterizing its variance is a more fair comparison to IS. We define the mini-batch IS estimator $X_{\text{IS}} \overset{\text{def}}{=} \frac{1}{k} \sum_{j=1}^k \rho_{z_j} \Delta_{z_j}$, where indices z_j are sampled uniformly from $\{1, \ldots, n\}$. This contrasts the indices $i_1, \ldots, i_k$ for

X_{BC} that are sampled proportionally to ρ_i.

We begin by characterizing the variance, under a fixed dataset B. For convenience, let $\mu_B = \mathbb{E}_\pi[\Delta|B]$. We characterize the sum of the variances of each component in the update estimator, which equivalently corresponds to normed deviation of the update from its mean,

$$\mathbb{V}(\Delta \mid B) \overset{\text{def}}{=} \operatorname{tr}\operatorname{Cov}(\Delta \mid B) = \textstyle\sum_{m=1}^{d} \operatorname{Var}(\Delta_m \mid B) = \mathbb{E}[\|\Delta - \mu_B\|_2^2 \mid B]$$

for an unbiased stochastic update $\Delta \in \mathbb{R}^d$. We show two theorems that BC-IR has lower variance than IS, with two different conditions on the norm of the update. We first start with more general conditions, and then provide a theorem for conditions that are likely only true in early learning.

Theorem B.1.1. *Assume that, for a given buffer B, $\|\Delta_j\|_2^2 > \frac{c}{\rho_j}$ for samples where $\rho_j \geq \bar{\rho}$, and that $\|\Delta_j\|_2^2 < \frac{c}{\rho_j}$ for samples where $\rho_j < \bar{\rho}$, for some $c > 0$. Then the BC-IR estimator has lower variance than the IS estimator: $\mathbb{V}(X_{\text{BC}} \mid B) < \mathbb{V}(X_{\text{IS}} \mid B)$.*

The conditions in Theorem B.1.1 preclude having update norms for samples with small ρ be quite large—larger than a number $\propto \frac{1}{\rho}$—and a small norm for samples with large ρ. These conditions can be relaxed to a statement on average, where the cumulative weighted magnitude of the update norm for samples with ρ below the median needs to be smaller than for samples with ρ above the mean.

We next consider a setting where the magnitude of the update is independent of the given state and action. We expect this condition to hold in early learning, where the weights are randomly initialized, and thus randomly incorrect across the state-action space. As learning progresses, and value estimates become more accurate in some states, it is unlikely for this condition to hold.

Theorem B.1.2. *Assume ρ and the magnitude of the update $\|\Delta\|_2^2$ are independent*

$$\mathbb{E}[\rho_j \|\Delta_j\|_2^2 \mid B] = \mathbb{E}[\rho_j \mid B]\ \mathbb{E}[\|\Delta_j\|_2^2 \mid B]$$

Then the BC-IR estimator will have equal or lower variance than the IS estimator: $\mathbb{V}(X_{\text{BC}} \mid B) \leq \mathbb{V}(X_{\text{IS}} \mid B)$.

These results have focused on variance of each estimator, for a fixed buffer, which provided insight into variance of updates when executing the algorithms. We would, however, also like to characterize variability across buffers, especially for smaller buffers. Fortunately, such a characterization is a simple extension on the above results, because variability for a given buffer already demonstrates variability due to different samples. It is easy to check that $\mathbb{E}[\mathbb{E}[\mu_{IR} \mid B]] = \mathbb{E}[\mu_{IS} \mid B] = \mathbb{E}_\pi[\Delta]$. The variances can be written using the law of total variance

$$\begin{aligned}
\mathbb{V}(X_{\text{BC}}) &= \mathbb{E}[\mathbb{V}(X_{\text{BC}} \mid B)] + \mathbb{V}(\mathbb{E}[X_{\text{BC}} \mid B]) = \mathbb{E}[\mathbb{V}(X_{\text{BC}} \mid B)] + \mathbb{V}(\mu_B) \\
\mathbb{V}(X_{\text{IS}}) &= \mathbb{E}[\mathbb{V}(X_{\text{IS}} \mid B)] + \mathbb{V}(\mu_B) \\
\implies \mathbb{V}(X_{\text{BC}}) - \mathbb{V}(X_{\text{IS}}) &= \mathbb{E}[\mathbb{V}(X_{\text{BC}} \mid B) - \mathbb{V}(X_{\text{IS}} \mid B)]
\end{aligned}$$

with expectation across buffers. Therefore, the analysis of $\mathbb{V}(X_{\text{BC}} \mid B)$ directly applies.

B.2 Additional Theoretical Results and Proofs

Lemma B.2.1. *Let $B_t = \{X_{t+1}, ..., X_{t+n}\}$ be the buffer of the most recent n transitions sampled by time $t+n$, which are generated sequentially from an irreducible, finite MDP with a fixed policy μ. We define $X_{\text{BC}}^{(t)}$ be the BCIR estimator for buffer B_t. If $\mathbb{E}_\pi[|\Delta|] < \infty$, then $\mathbb{E}[X_{\text{BC}}^{(t)}] = \mathbb{E}_\pi[\Delta]$.*

Proof. Let $X_t = (S_t, A_t, R_{t+1}, S_{t+1})$ be a transition and $\{B_t\}_{t\in\mathbb{N}}$ be the sequence of buffers that are observed, each containing n consecutive transitions.

Using the law of iterated expectations,

$$\mathbb{E}\left[X_{\text{BC}}^{(t)}\right] = \mathbb{E}\left[\mathbb{E}[X_{\text{BC}}^{(t)}|B_t]\right]$$

where the outer expectation is over the stationary distribution of B_t and the inner expectation is over the sampling distribution of IR from the buffer B_t.

Using the definition of $X_{\text{BC}}^{(t)}|B_t$, we have that

$$\begin{aligned}
\mathbb{E}[X_{\text{BC}}^{(t)}|B_t] &= \bar{\rho}\sum_{i=1}^{n} \Delta_i \frac{\rho_i}{\sum_{i=1}^{n}\rho_i} \\
&= \frac{1}{n}\sum_{i=1}^{n} \rho_i \Delta_i
\end{aligned}$$

Next, the stationary distribution of B_t is given by $d(B_t) = Pr(B_t = (x_{t+1}, ..., x_{t+n})) =$
$d_X(x_t)p(x_{t+1}|x_t)...p(x_{t+n}|x_{t+n-1})$, where d_X is the stationary distribution of X_t. We can verify directly by checking that for all $B' = (x_2, ..., x_{n+1})$

$$\sum_B d(B)p(B'|B) = d(B')$$

where $B = (x_1, ..., x_n)$

To see this, first note that $p(B'|B) = p(x_{n+1}|x_n)\mathbf{1}(B, B')$ where $\mathbf{1}(B, B')$ is equal to 1 if the states $(x_2, ..., x_n)$ in B match the states $(x_2, .., x_n)$ in B'. In other words, the first $n-1$ transitions in B' must match the last $n-1$ transitions in B for $p(B'|B)$ to be positive. Next, fixing B',

$$\begin{aligned}
&\sum_B d(B)p(B'|B) \\
&= \sum_{x_1,...,x_n} d_\mu(x_1)p(x_2|x_1)...p(x_n|x_{n-1})p(x_{n+1}|x_n)\mathbf{1}(B, B') \\
&= \sum_{x_1} d_\mu(x_1)p(x_2|x_1)...p(x_n|x_{n-1})p(x_{n+1}|x_n) \quad \text{since } (x_2, ..., x_n) \text{ have to match} \\
&= d_\mu(x_2)p(x_3|x_2)...p(x_n|x_{n-1})p(x_{n+1}|x_n) \\
&= d(B')
\end{aligned}$$

which verifies the expression for the stationary distribution of B_t.

Continuing from before, we expand the expectation as:

$$\begin{aligned}
&\mathbb{E}\left[\frac{1}{n}\sum_{t=1}^{n} \rho_t \Delta_t\right] \\
&= \sum_{x_1,...x_n} d_X(x_1)p(x_2|x_1)...p(x_n|x_{n-1}) \left(\frac{1}{n}\sum_{t=1}^{n} \rho_t \Delta_t\right) \\
&= \sum_{\substack{s_1,a_1,r_2,s_2,..., \\ s_n,a_n,r_{n+1},s_{n+1}}} d_\mu(s_1) \left(\prod_{i=1}^{n} \mu(a_i|s_i)p(s_{i+1}, r_{i+1}|s_i, a_i)\right) \left(\frac{1}{n}\sum_{t=1}^{n} \rho_t \Delta_t\right) \\
&= \frac{1}{n}\sum_{t=1}^{n} \sum_{\substack{s_1,a_1,r_2,s_2,..., \\ s_n,a_n,r_{n+1},s_{n+1}}} d_\mu(s_1) \left(\prod_{i=1}^{n} \mu(a_i|s_i)p(s_{i+1}, r_{i+1}|s_i, a_i)\right) (\rho_t \Delta_t).
\end{aligned}$$

Next, by taking the sums over $(s_1, a_1, ...r_{n+1}, s_{n+1})$ within the products to make the summands depend only on the variables being summed over, we get

$$= \frac{1}{n}\sum_{t=1}^{n}\sum_{s_1} d_\mu(s_1) \sum_{a_1,r_2,s_2} \mu(a_1|s_1)p(s_2,r_2|s_1,a_1) \sum_{a_2,r_3,s_3} \mu(a_2|s_2)p(s_3,r_3|s_2,a_2)...$$
$$\sum_{a_t,r_{t+1},s_{t+1}} \mu(a_t|s_t)p(s_{t+1},r_{t+1}|s_t,a_t)\,(\rho_t\Delta_t)$$
$$\sum_{\substack{a_{t+1},r_{t+2},s_{t+2},...,\\ s_n,a_n,r_{n+1},s_{n+1}}} \prod_{i=t+1}^{n} \mu(a_i|s_i)p(s_{i+1},r_{i+1}|s_i,a_i)$$
$$= \frac{1}{n}\sum_{t=1}^{n}\sum_{s_1} d_\mu(s_1) \sum_{a_1,r_2,s_2} \mu(a_1|s_1)p(s_2,r_2|s_1,a_1) \sum_{a_2,r_3,s_3} \mu(a_2|s_2)p(s_3,r_3|s_2,a_2)...$$
$$\sum_{a_t,r_{t+1},s_{t+1}} \mu(a_t|s_t)p(s_{t+1},r_{t+1}|s_t,a_t)\,(\rho_t\Delta_t)\,.$$

This followed since the third line is summing over the probability of all trajectories starting from s_{t+1} and thus is equal to 1. Next, we note that, if C is a constant that does not depend on s_1, a_1, r_2, then $\sum_{s_1,a_1,r_2} d_\mu(s_1)\mu(a_1|s_1)p(s_2,r_2|s_1,a_1)C = d_\mu(s_2)C$ since $d_\mu(s_2)$ is the stationary distribution (if we additionally assume $p(s_2,r_2|s_1,a_1) = p(s_2|s_1,a_1)p(r_2|s_1,a_1)$ or equivalently that rewards depend only on state and action).

Continuing from before, by reordering the sums we have and repeatedly using the above note,

$$= \frac{1}{n}\sum_{t=1}^{n}\sum_{s_2} \underbrace{\sum_{s_1,a_1,r_2} d_\mu(s_1)\mu(a_1|s_1)p(s_2,r_2|s_1,a_1)}_{d_\mu(s_2)} \sum_{a_2,r_3,s_3} \mu(a_2|s_2)p(s_3,r_3|s_2,a_2)...$$
$$\sum_{a_t,r_{t+1},s_{t+1}} \mu(a_t|s_t)p(s_{t+1},r_{t+1}|s_t,a_t)\,(\rho_t\Delta_t)$$
$$= \frac{1}{n}\sum_{t=1}^{n}\sum_{s_2} d_\mu(s_2) \sum_{a_2,r_3,s_3} \mu(a_2|s_2)p(s_3,r_3|s_2,a_2)...$$
$$\sum_{a_t,r_{t+1},s_{t+1}} \mu(a_t|s_t)p(s_{t+1},r_{t+1}|s_t,a_t)\,(\rho_t\Delta_t)$$
$$= ... \quad \text{(Repeating the same process)}$$
$$= \frac{1}{n}\sum_{t=1}^{n} \sum_{s_t,a_t,r_{t+1},s_{t+1}} d_\mu(s_t)\mu(a_t|s_t)p(s_{t+1},r_{t+1}|s_t,a_t)\,(\rho_t\Delta_t)\,.$$

Recall that $\Delta_t = \Delta(s_t, a_t, r_{t+1}, s_{t+1})$ is a function of the transition so we cannot simplify further.

Finally,

$$\begin{aligned}
&= \frac{1}{n}\sum_{t=1}^{n}\sum_{s_t,a_t,r_{t+1},s_{t+1}} d_\mu(s_t)\mu(a_t|s_t)p(s_{t+1},r_{t+1}|s_t,a_t)\left(\frac{\pi(a_t)}{\mu(a_t)}\Delta_t\right)\\
&= \frac{1}{n}\sum_{t=1}^{n}\sum_{s_t,a_t,r_{t+1},s_{t+1}} d_\mu(s_t)\pi(a_t|s_t)p(s_{t+1},r_{t+1}|s_t,a_t)\Delta_t\\
&= \frac{1}{n}\sum_{t=1}^{n}\mathbb{E}_\pi[\Delta]\\
&= \mathbb{E}_\pi[\Delta]
\end{aligned}$$

□

Theorem 9.3.2 Let $B_t = \{X_{t+1}, ..., X_{t+n}\}$ be the buffer of the most recent n transitions sampled by time $t+n$, which are generated sequentially from an irreducible, finite MDP with a fixed policy μ. Define the sliding-window estimator $X_t \overset{\text{def}}{=} \frac{1}{T}\sum_{t=1}^{T} X_{\text{BC}}^{(t)}$. Then, if $\mathbb{E}_\pi[|\Delta|] < \infty$, then X_T converges to $\mathbb{E}_\pi[\Delta]$ almost surely as $T \to \infty$.

Proof. Let $X_t = (S_t, A_t, R_{t+1}, S_{t+1})$ be a transition. Then the sequence $\{X_t\}_{t\in\mathbb{N}}$ forms an irreducible Markov chain as there is positive probability of eventually visiting any X' starting from any X since this is true for states S' and S in the original MDP (by irreducibility).

Let $\{B_t\}_{t\in\mathbb{N}}$ be the sequence of buffers that are observed. This also forms an irreducible Markov chain by the same reasoning as above since $\{X_t\}_{t\in\mathbb{N}}$ is irreducible. Additionally, the sequence of pairs $\{(X_{\text{BC}}^{(t)}, B_t)\}_{t\in\mathbb{N}}$ is an irreducible Markov chain.

Using the ergodic theorem (theorem 4.16 in (**levin2017markov**)) on $\{(X_{\text{BC}}^{(t)}, B_t)\}_{t\in\mathbb{N}}$ with the projection function $f(x,y) = x$, we have that

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T} X_{\text{BC}}^{(t)} = \mathbb{E}\left[X_{\text{BC}}^{(t)}\right]$$

where the expectation is over the joint stationary distribution of $(X_{\text{BC}}^{(t)}, B_t)$.

Using Lemma B.2.1 we can show that $\mathbb{E}\left[X_{\text{BC}}^{(t)}\right] = \mathbb{E}_\pi[\Delta]$, completing the proof.

$\square$

B.3 Expanded Empirical Results

B.3.1 Markov Chain

This section contains the full set of markov chain experiments using several different policies. Results can be found in figure 9.3 and figure B.2. See figure captions for more details.

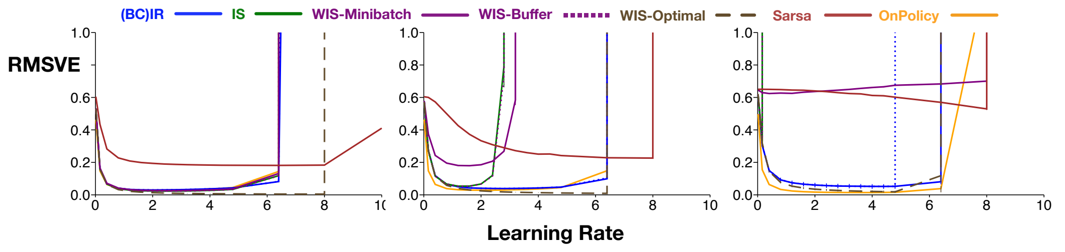


Figure B.1: Sensitivity curves for Markov Chain experiments with policy action probabilities [left, right] **left** $\mu = [0.5, 0.5], \pi = [0.1, 0.9]$; **center** $\mu = [0.9, 0.1], \pi = [0.1, 0.9]$; **right** $\mu = [0.99, 0.01], \pi = [0.01, 0.99]$.

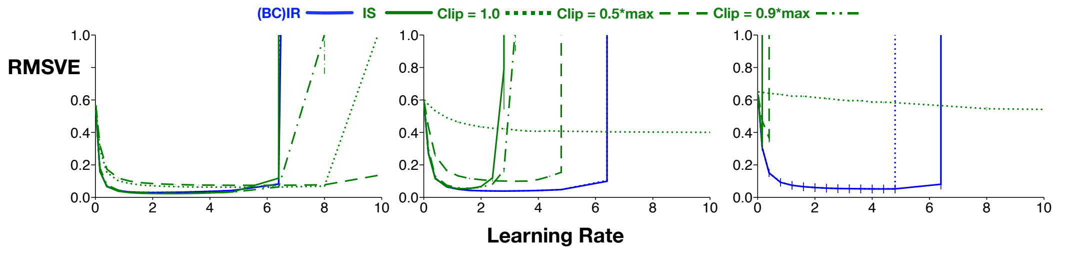


Figure B.2: Learning rate sensitivity plots for V-Trace (with the same settings as Figure 9.3). Three clipping parameters were chosen, including 1.0, 0.5 $\rho_{\max}$ and 0.9 $\rho_{\max}$, where $\rho_{\max}$ is the maximum possible IS ratio. For 1.0 $\rho_{\max}$, updates under V-trace become exactly equivalent to IS.

B.3.2 Continuous Four Rooms

The continuous four rooms environment is an 11x11 2d continuous world with walls setup as the original four rooms environment grid world. The agent is a

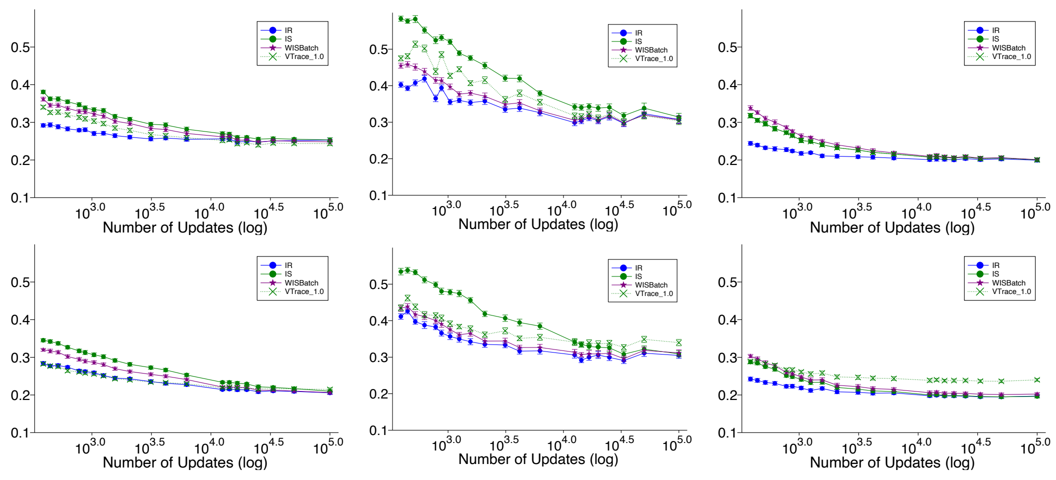


Figure B.3: SGD: Target Policy: **Top**: persistent down, **Bottom** favored down. Behavior Policy: **left** State Variant **center** State Weight Variant **right** Uniform. Sample efficiency plots.

circle with radius 0.1, and the state consists of a continuous tuple containing the x and y coordinates of the agent's center point. The agent takes an action in one of the 4 cardinal directions moving $0.5 \pm \mathbb{U}(0.0, 0.1)$ in that directions and random drift in the orthogonal direction sampled from $\mathbb{N}(0.0, 0.01)$. The simulation takes 10 intermediary steps to more accurately detect collisions.

We use three behavior policies in our experiments. **Uniform:** the agent selects all actions uniformly, **State Variant:** the agent selects all actions uniformly except in pre-determined subsections of the environment where the agent will take down with likelihood 0.1 and the rest distributed evenly over the other actions, **State Weight Variant:** the agent selects all actions uniformly except in pre-determined subsections where the pmf is defined randomly. We also use two target policies. **Persistent Down:** where the agent always takes the down action, **Favored Down:** where the agent takes the down action with likelihood 0.9 and uniformly among the other actions with likelihood 0.1. We use a cumulant function which indicates collision with a wall and a termination function which terminates on collision and is 0.9 otherwise for all value functions. We present results using SGD and RMSProp over many algorithms and parameter settings in figures B.3, B.4, B.5, and B.6.

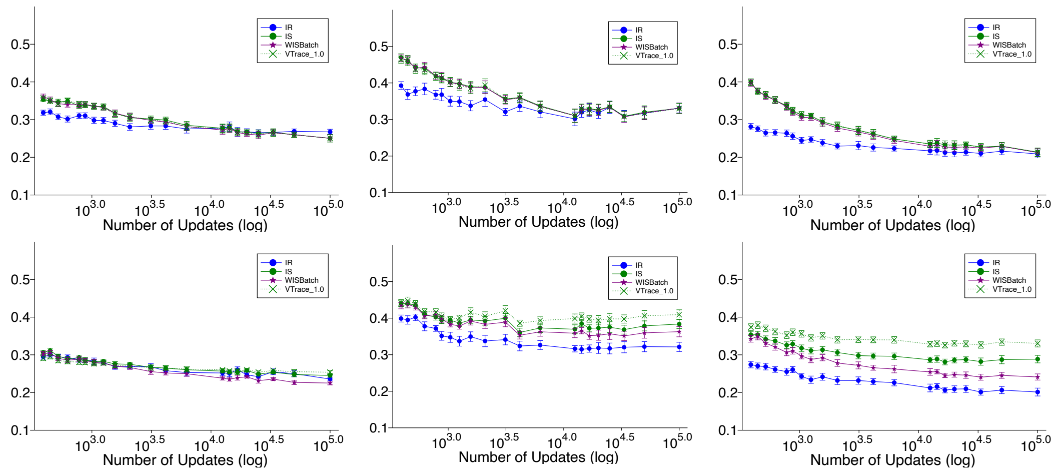


Figure B.4: RMSProp Target Policy: **Top**: persistent down, **Bottom** favored down. Behavior Policy: **left** State Variant **center** State Weight Variant **right** Uniform. Sample efficiency plots.

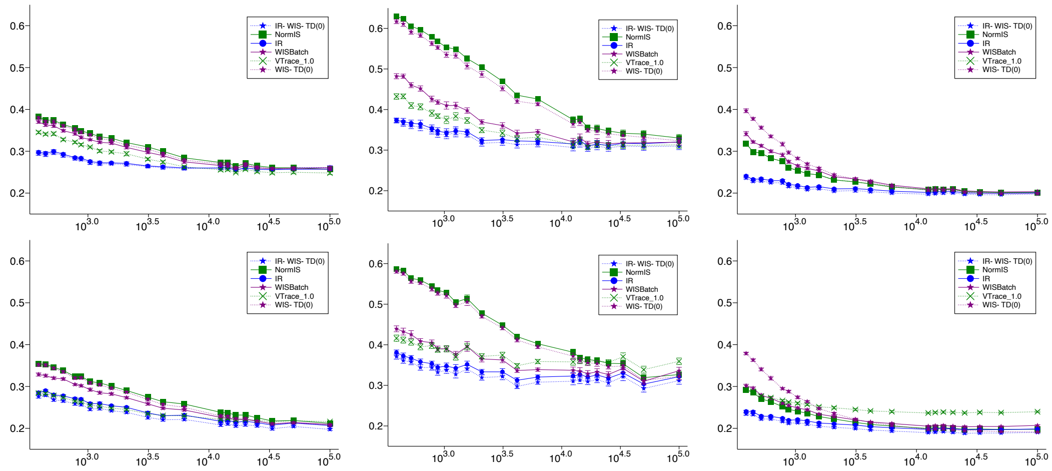


Figure B.5: Incremental Experiments Target Policy: **Top**: persistent down, **Bottom** favored down. Behavior Policy: **left** State Variant **center** State Weight Variant **right** Uniform. Sample efficiency plots.

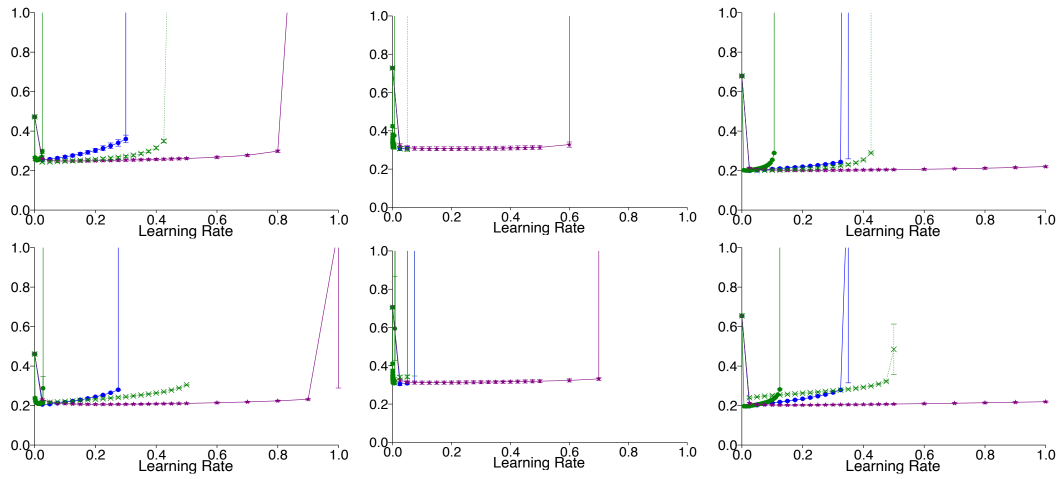


Figure B.6: SGD: Target Policy: **Top**: persistent down, **Bottom** favored down. Behavior Policy: **left** State Variant **center** State Weight Variant **right** Uniform. Learning Rate Sensitivity