Representation and General Value Functions

by

Craig Sherstan

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

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

Research in artificial general intelligence aims to create agents that can learn from their own experience to solve arbitrary tasks in complex and dynamic settings. To do so effectively and efficiently, such an agent must be able to predict how its environment will change both dependently and independently of its own actions. General value functions (GVFs) are one approach to representing such relationships. A single GVF poses a predictive question defined by three components: a behavior (policy), a prediction timescale, and a predicted signal (cumulant). Estimated answers to these questions can be learned efficiently from the agent's own experience using temporal-difference learning methods. The agent's collection of GVF questions and corresponding answers can be viewed as forming a predictive model of the agent's interaction with its environment. Ultimately, such a model may enable an agent to understand its environment and make decisions therein.

Although GVFs are a promising approach, current understanding of their construction and use remains limited. This dissertation explores several aspects of GVF usage and representation and can be grouped into two areas of research. The first area concerns what information can be represented by GVFs. We suggest that the GVF format might be used by an agent for performing introspection or self-examination. Specifically, we propose using internally generated signals as cumulants for introspective GVFs and argue that such predictions can enhance an agent's state representation. We explore the behavior of several introspective signals in various domains. We then present a new algorithm that uses a series of GVFs to directly estimate the variance of the *return*—the sum of future signal. The variance of the return can be viewed as one such introspective signal capable of enhancing an agent's decision making process.

This dissertation's second focus is on improving the representations used to estimate the answers to GVFs. Value functions can be factored into two components, one representing the signal of interest and the other representing the dynamics of the environment—the *successor representation*. We show that in the context of a constructive GVF framework, in which new GVF targets are identified over time, using the successor representation can speed up learning of newly added targets. Next, we introduce Γ-nets, which enable a single GVF estimator to make predictions for any fixed timescale within the training bounds, improving the tractability of learning and representing vast numbers of predictions. Finally, we present investigations into how GVFs, including Γ-nets, can be used as auxiliary tasks to improve representation learning.

In summary, this dissertation provides new perspectives, algorithms and empirical evaluations that we believe will benefit the broader work in predictive approaches to artificial general intelligence. Specifically, our work on GVF representations provides a direction for future research on the topic of introspection, and the work on representing GVFs demonstrates methods for making learning and representing large collections of GVFs more tractable for real-world problems.

Preface

This dissertation is written in a mixed format using published papers as chapters with minimal modification.

- The ideas underlying Chapters 3 and 4 were originally posed in my MSc thesis:

  Sherstan, C. (2015). *Towards Prosthetic Arms as Wearable Intelligent Robots* (Master's thesis, University of Alberta).

  These ideas were expanded and the chapters include work from the following publications:

  1. Sherstan, C., Machado, M. C., White, A., & Pilarski, P. M. (2016). Introspective Agents: Confidence Measures for General Value Functions. In *Artificial General Intelligence (AGI)* (pp. 258–261). New York, New York, USA: Springer International Publishing
     I was the principal author and sole experimentalist.
  2. Pilarski, P. M. & Sherstan, C. (2016). Steps Toward Knowledgeable Neuroprostheses. *IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)* (p. 220). Singapore
     I contributed conceptual ideas and some text.
  3. Sherstan, C., Machado, M. C., Travnik, J., White, A., Vasan, G., & Pilarski, P. M. (2017). Confident Decision Making with General Value Functions. *Reinforcement Learning and Decision Making (RLDM)*. Ann Arbor, USA
     I was the principal author and experimentalist.
  4. Günther, J., Kearney, A., Dawson, M. R., Sherstan, C., & Pilarski, P. M. (2018). Predictions, Surprise, and Predictions of Surprise in General Value Function Ar-

chitectures. *AAAI Fall Symposium - Reasoning and Learning in Real-World Systems for Long-Term Autonomy* (pp. 22–29). Arlington, USA
I contributed ideas and text.

Chapter 3 also contains unpublished work done in collaboration with Adam White, Marlos C. Machado, and Patrick M. Pilarski.

- Chapter 5 is based on

  Sherstan, C., Ashley, D. R., Bennett, B., Young, K., White, A., White, M., & Sutton, R. S. (2018). Comparing Direct and Indirect Temporal-Difference Methods for Estimating the Variance of the Return. *Conference on Uncertainty in Artificial Intelligence (UAI)* (pp. 63–72). Monterey, USA

  I was the principal author and experimentalist. The original algorithm was posed by Kenny Young and Richard S. Sutton.

- Chapter 6 is based on

  Sherstan, C., Machado, M. C., & Pilarski, P. M. (2018). Accelerating Learning in Constructive Predictive Frameworks with the Successor Representation. *IEEE International Conference on Robots and Systems (IROS)* (pp. 2997–3003). Madrid, Spain

  I was the principal author and sole experimentalist. The original idea for this work was developed in collaboration with Marlos C. Machado.

- Chapter 7 was conceived while I was an intern at Cogitai. The chapter is based on

  Sherstan, C., Dohare, S., MacGlashan, J., Günther, J., & Pilarski, P. (2020). Gamma Nets: Generalizing Value Estimation over Timescale. *AAAI Conference on Artificial Intelligence* (pp. 5717–5725). New York, USA

  For these works I was the principal author and experimentalist.

- Section 8.2, auxiliary tasks with Γ-nets, is unpublished work conducted with undergraduate student Jesse Farebrother under my supervision and direction.

- Section 8.3 is work that was started while I was an intern at Borealis AI. It was presented at the ALA workshop at AAMAS 2020. For this work I conceived the idea, carried out the experiments and completed the majority of writing.

  Sherstan, C., Kartal, B., Hernandez-Leal, P., & Taylor, M. E. (2020). Work in Progress: Temporally Extended Auxiliary Tasks. *Adaptive and Learning Agents (ALA) Workshop at AAMAS*. Auckland, New Zealand (Virtual)

Acknowledgements

These acknowledgements are thanks to the people who kept me sane and kept me going. I won't pretend this was easy, it wasn't, but all of you helped me finish.

First and foremost I thank my wife, Jen, and my kids, Nika and Kai. You have been a constant support and reminder that there are more important things in life than a PhD. You are the best distractions.

In alphabetical order by first name:

- Adam White, I always enjoyed the chance to work with you. Your interaction and encouragement has been a tremendous help over the years.
- Alex Kearney, thank you for treating me like I had something worth contributing.
- Anna Koop, you always manage to make me feel good about myself.
- Marlos C. Machado, you have been a constant friend and source of encouragement.
- Mike Bowling, thank you for saying "this is normal, you're doing fine."
- Patrick M. Pilarski, thank you for always being excited about my research.

Contents

List of Tables

List of Figures

Glossary

agent An entity capable of making decisions and taking actions.

auxiliary task For the purposes of this dissertation an auxiliary task is any network output whose sole purpose is to provide gradient for training a representation network that is shared by an agent's policy. See Chapter 8.

BLINC Bionic Limbs for Improved Natural Control Lab, University of Alberta, Edmonton, Canada.

certainty For the puroses of Part I of this dissertation this is defined as the width of a distribution over a prediction's output. See Section 3.1.

confidence For the purposes of Part I of this dissertation this is defined as an agent's own estimation of the probability that its choice is the correct one, given all its current knowledge. See Section 3.1.

cumulant The signal by the which the return of a GVF is defined—the sum of future cumulants.

environment Everything outside of the agent. Actions are sent to the environment and the agent receives rewards and observations from the environment. The environment should not be seen as synonymous with the physical world or the agent with a robot's body. The boundary between environment and agent is more likely to be within the mind of the agent itself.

function approximation Function approximation involves representing a complex or unknown target function using a simpler well-known function.

general value function (GVF) General value functions pose a predictive question defined by the syntax of value functions used in reinforcement learning. A GVF question is defined by three components: 1) policy, 2) cumulant, 3) timescale.

generalization The ability to make inferences about things that are *similar* to what has been seen before, but not the same.

intelligent An intelligent agent is one that is able to learn about itself, its environment,and the interaction between the two and adapt its actions to improve its ability to accomplish a goal.

model In contrast to the strict definitions often used in RL, we use the term *model* to mean any representation that captures some aspects of the dynamics of the agent, its environment, and their interaction.

off-policy Learning about a target policy from data collected under a different behavior policy.

offline Offline machine learning methods separate the collection of samples from the learning from those samples. That is, there is a sample collection phase, followed by a learning phase.

on-policy Learning about the behavior policy from data collected under that policy.

online Online machine learning methods are those that learn from each data sample as it occurs.

policy A policy is a way of behaving. It is a deterministic or stochastic mapping from a state to an action.

Reinforcement Learning (RL) A problem setting in which an agent learns directly from experience how to behave, so as to maximize long-term reward.

return The sum of future rewards or cumulants. The agent's goal is to learn a policy to maximize the return. In practice, various proxy-returns are used instead.

reward A scalar signal that evaluates an agent's performance at each timestep.

RLAI Reinforcement Learning and Artificial Intelligence lab, University of Alberta, Edmonton, Canada.

Robot Operating System (ROS) An open-source framework for robotics software.

state The configuration of the universe and everything in it at a given moment in time. In real-world scenarios an agent has limited observation and must rely on its own, internal representation of state, which is an approximation.

Successor Representation (SR) A method for factoring value estimation. The SR captures the dynamics of the environment with respect to a given policy and timescale independent of any reward or cumulant.

Temporal-Difference (TD) A family of algorithms used in reinforcement learning that learn to estimate value incrementally by updating towards an approximate return composed of environment samples and bootstrapped estimates.

timescale A GVF has an associated prediction timescale, which can be roughly thought of as the prediction length.

timestep (ts) Typically one timestep is one state-action-state transition tuple. Timestep units are denoted by ts.

value The expected sum of future rewards (or cumulant) starting from a state and following a particular policy.

Notation

S, R, A — random variables are indicated by capital letters, while lower-case letters are used to indicate specific instances of a variable. Lower-case letters are also used for functions and scalars.
$\mathcal{S}, \mathcal{A}$ — calligraphics are used to indicate sets
$\mathbf{w}, \phi, \mathbf{e}$ — vectors are indicated in bold
$\mathbb{R}^n$ — vector of n real-valued numbers.
$^\top$ — transpose operator
$\mathbb{E}$ — expectation
I — identity matrix
$\mathcal{N}$ — Normal (Gaussian) distribution

Frequently used variables

A, a — action. An action taken or available to an agent. See Figure 2.1.
α — step-size. Used to control how big of a step an iterative update (such as Gradient Descent or Stochastic Gradient Descent) takes to adjusting its prediction. See Equation 2.6.
C, c — cumulant. The signal accumulated by the GVF. See Section 2.4.
δ — temporal-difference error. The difference in the current prediction and prediction target. See Equation 2.5.
G — return. Summation of future cumulant. See Section 2.4.
γ — temporal discounting factor. Used in value functions to adjust emphasis between near-term and far-term signals. See Equation 2.2.
λ — bootstrapping parameter. Controls the trade off between bootstrapping and Monte Carlo backups in the TD update. See Equation 2.9.
ϕ — feature vector. A representation of the state. See Section 2.3.
Q — state-action value, the expected return under a specified policy starting from a state s and taking action a and then following the specified policy. See Equation 2.4.
R, r — reward. A signal used to evaluate a goal-seeking agent's actions. See Figure 2.1.
ρ — importance sampling ratio. This is the ratio between the likelihood of taking action a in state s under a target policy and a behavior policy. See Equation 2.11.
S, s — state. The configuration of the agent and the agent's environment. State may not be directly knowable. See Figure 2.1.
t — time.
τ — expected number of timesteps. This variable corresponds to the timescale of a value function prediction. See Equation 2.13.
V — state value, the expected return under a specified policy starting from a state s. See Equation 2.3).
$\mathbf{w}$ — learned model parameters. See Chapter 2.
X — used to denote an arbitrary signal.
$\mathbf{z}$ — trace vector. A vector used to track state visitations and assign credit. See Equation 2.10.

Chapter 1

Introduction

The real-world is large and complex and cannot be known fully by either robots or humans, nor can a robot be preprogrammed to deal with all situations or tasks. My research is motivated towards creating robots capable of *continual learning*—the incremental acquisition of increasingly complex knowledge and skill over time, directly from a robot's own sensorimotor experience, with no singular end in mind (Ring, 1994). Continual learning robots would be generally useful and would benefit broad aspects of human society in ways that are presently limited to works of fiction. Such robots could help humans in their daily lives, cleaning up around the house and preparing supper. Such robots might perform complex life-saving medical operations, or autonomously construct houses, or repair spacecraft. The creation of robots such as these is the motivation for the work presented in this dissertation.

In reinforcement learning (RL) an agent, virtual or robotic, learns how to behave in its environment so as to accomplish a goal. As the agent interacts with its environment it receives feedback in the form of a scalar reward signal. Better behavior produces larger reward and the agent learns to adapt its behavior to maximize the amount of cumulative reward it receives in the future. It is important to note that in the context of continual learning we are not as concerned that an agent achieves optimal performance on any one task, but rather that it has the algorithmic mechanisms needed to continually improve its performance on all tasks that it undertakes.

Broadly, my research focuses on the incremental construction of *models* in the context of RL, one of the key open problems in artificial intelligence (AI). While those with backgrounds in RL may associate the term *model* with the one-step transition and reward models, here I use the term in a broader, and arguably more common, sense. For the purposes of this

dissertation a model is a representation of a system and its dynamics and can be used to predict how the system will respond to input (including time). In AI a model allows an agent to represent the world, itself and the interactions between the two and to predict how this system, both the environment and the agent itself, will respond due to the agent's own behavior or other factors in the system. Ultimately, models enable agents to make decisions.

A complete model is one that reflects all the dynamics of the environment, accurately predicting all aspects of the future in so far as it is predictable. With complete models an agent can theoretically make optimal decisions. In simple settings a complete model might be provided to the agent by an expert. In complex settings agents must instead learn their models over time, based on their own experience. The agent must therefore make decisions with incomplete and inaccurate models that change. This creates an interesting cycle in that the agent's model affects the decisions it makes, which in turn affects the experiences it observes, which in turn affects how it refines its model, which again affects the decisions it makes and so on. This process of iterative model construction is a key component to continually learning robots.

Robots (and humans) are bound by resources such as time, energy, limited observation, memory and computation. They must make decisions quickly as the real-world continues to move forward without waiting for their choices. Thus, I desire algorithms that are light weight, which can scale indefinitely with experience and that allow rapid online learning and adaptation. Further, I desire techniques that are not dependent on human intervention or design, but rather I look for general solutions and principles—views that have been shaped by many of my colleagues and teachers.

General value functions (GVFs) (Sutton, Modayil, et al., 2011), which are described in detail in Section 2.4, are one framework for representing predictive models. A GVF asks a single predictive question about a prediction target, the *cumulant*, with respect to a way of behaving, a *policy*, over a prediction timescale. For example, consider a simple mobile robot, GVF questions could include:

> "If I drive forward, how much current will my motors draw over the next 3 seconds?"
>
> "If I drive forward, how long until my bump sensor is activated?"

A GVF is grounded directly in the agent's own sensorimotor stream and action space, not depending on any privileged signals or representations interpreted by a human. A GVF can be treated as an experiment that an agent can perform on its own sensorimotor experience. The agent can make a prediction, "If I drive forward my motors will use 10 amps in the next 3 seconds." The agent can then drive forward for 3 seconds and validate or invalidate its prediction, using the error to improve its estimate for next time.

For a continually learning agent using a GVF-based model we want an algorithm that incrementally constructs the agent's model as it gains more experience. While this process of incremental construction is still unsolved, we can imagine desirable properties of such a process. Newly added GVFs should leverage existing GVFs whenever appropriate. A GVF might predict the output of another GVF or be used as state information to inform another GVF's own predictions. Thus, we desire a GVF-based model to be hierarchical, growing from simpler predictions to increasingly complex and abstract ones. Additionally, it is desirable that models be compositional. That is, the GVF model should be able to combine different predictions in such a way that it can answer questions for which it does not have an explicit representation. This is increasingly important in large domains to ensure compact representations and sample efficiency. For example, imagine that I am planning a holiday for my family to go to Kyoto, Japan. I have travelled extensively and have even been to Kyoto, but not with my family. Rather than having to relearn all aspects of trip planning, I am able to reuse what I have learned before to plan for this new trip. In the RL literature this is referred to as *transfer*—reusing what was learned in one situation for another. A specific example of compositionality in GVFs is demonstrated in Chapter 6.

A GVF specifies a question and various representations and algorithms are used to learn an estimate for the answer. Throughout this dissertation we use temporal-difference (TD) learning methods (Sutton and Barto, 2018), which are described in Section 2.2, to learn these estimates. TD methods have many appealing properties: they can be computationally efficient and computed *online*—learning can occur continually without having to wait until a task has completed. This is achieved by *bootstrapping*—learning from estimates. TD methods allow GVFs to monitor their own accuracy and self-correct. Thus, TD learning and GVFs provide a way of grounding the agent's model in its own experience and sensorimotor stream.

My dissertation makes contributions to the topic of representation and GVFs. The con-

tributions of Part I look at what sorts of questions can be represented with GVFs. In Part II, I present contributions relating to how to represent the answers posed by GVFs. In Part III, we consider how GVFs can be used to drive representation learning. My dissertation's contributions are summarized in the following section.

1.1 Contributions

Part I - Questions: Representation with GVFs

- **Introspective State Representations** (Chapter 3). We propose that an agent's state representation include measures that describe its internal state and statistics of its learning experience and sensorimotor stream. We suggest that some such measures can be captured by GVFs and, further, that such measures might themselves be predicted by GVFs. This perspective motivates the remainder of the work in Part I.

- **Evaluation of Introspective Measures** (Chapter 4). We provide empirical demonstrations of the behavior of various GVF-based introspective measures on simulation and robot domains. These examples suggest how such measures might provide the agent with information currently lacking in its representation, without which it cannot be expected to make optimal decisions.

- **Estimating the Variance of the Return Using GVFs** (Chapter 5). We present a novel algorithm, Direct Variance TD (DVTD), for directly estimating the variance of the *return* by using a series of GVFs. We empirically evaluate DVTD, against an indirect method, Variance TD (VTD) and find that in all tested cases DVTD performs just as well and sometimes better. Additionally, we provide theoretical bounds on the variance estimate as a function of the error in the value estimate.

Part II - Answers: Representing GVFs

- **Accelerating GVF Learning Using the Successor Representation** (Chapter 6). Here we assume that a process of GVF construction exists. In this setting we demonstrate how the successor representation (SR), which captures the dynamics of the environment, can be used to accelerate learning of incrementally added GVFs. This is validated experimentally in both a tabular grid-world and on a robotic arm.

- **Generalizing Value Estimation Over Timescale** (Chapter 7). We introduce a novel method, Γ-nets (Gamma-nets), which generalizes value estimation over timescale. The method allows a single network to make GVF predictions for any fixed timescale within the training bounds. We show that Γ-nets can be effectively learned in the policy evaluation setting on simulation and robotic domains. We consider various implementation approaches for Γ-nets and empirically evaluate the effect they have on prediction performance.

Part III - Driving Representation Learning with GVFs

- **Γ-nets as Auxiliary Tasks (Section 8.2).** We demonstrate that Γ-nets can improve policy performance when used as auxiliary tasks on several Atari games.

- **Temporal-Difference Auxiliary Tasks (Section 8.3).** We examine the effect that the temporal scale of a GVF has on representation learning when used as an auxiliary task. We introduce the temporal-difference auto encoder (TD-AE) and evaluate its effect on the performance of agents on the ViZDoom environment using the A2C algorithm. In our experiments, we found no direct relationship between the temporal scale and agent performance. However, we do find that adding such auxiliary tasks can increase the algorithm's robustness to the sequence length parameter used by A2C.

As stated in the opening paragraph of this chapter, my goal is to create robots that are capable of continual learning. More specifically, the research presented in this dissertation has been motivated towards the creation of intelligent prosthetic limbs—limbs which can adapt to their user and the environment around them and improve the user's ability to perform everyday tasks. As such, while much of the validating experiments presented in this dissertation are done in simulation, we will often include experiments on robot arms which are intended to validate the ideas in a more realistic, prosthetics-inspired setting.

The idea that predictions are a key component to AI systems is not a new one, in fact it is quite popular. GVFs present one possible predictive architecture. They have many characteristics that make them appealing, as we have already discussed. However, they remain in the research stage with many open questions. The work in my dissertation provides new knowledge about the relationship between GVFs and representation and will enable further progress towards incorporating GVFs into real-world robotics in useful ways.

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement learning (RL) is the problem of how an agent can learn from a scalar reward signal. The RL problem is illustrated in Figure 2.1. At time, t, the agent receives some observation of state, S_t, from the environment, takes some action, A_t and receives a reward, R_{t+1} and the next observation S_{t+1}. The agent's task is to learn how to maximize the sum of its future rewards. It is important to distinguish that RL is a problem setting, not a set of solution methods.

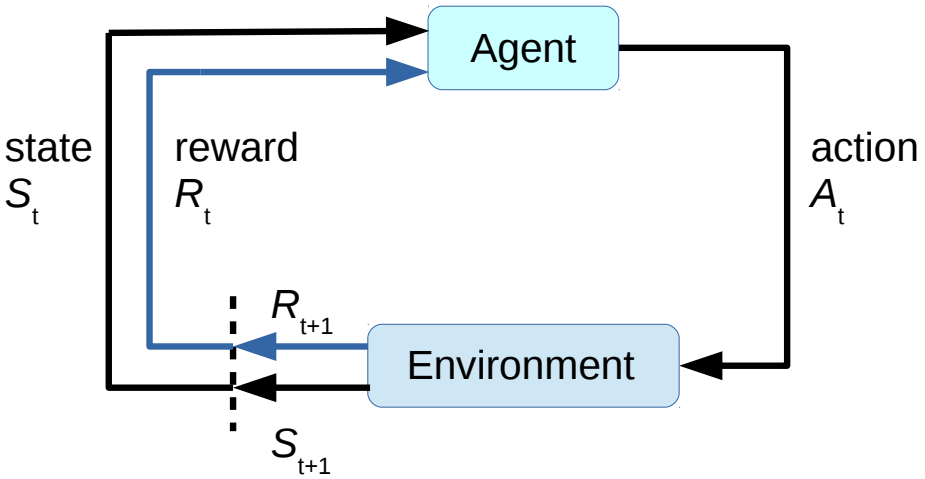


Figure 2.1: The Reinforcement Learning setting.

It is convenient to consider the RL problem as a Markov Decision Process (MDP) that is defined by the tuple $< \mathcal{S}, \mathcal{A}, P, r >$, where $\mathcal{S}$ is the finite set of states, $\mathcal{A}$ the set of actions the agent can take, $p \doteq (s'|s, a)$ gives state transition probabilities, $r(s, a, s') \in \mathbb{R}$ the reward function. This translates to the following interaction. At each timestep t the

agent observes the state of the world $S_t \in \mathcal{S}$ and chooses an action $A_t \in \mathcal{A}$ according to its policy $\pi(a|S_t) \in [0, 1]$, that gives the probability of choosing each action, $a \in \mathcal{A}$ given the state S_t. The environment transitions to a new state S_{t+1} according to $p(S_{t+1}|S_t, A_t)$ and receives a reward R_{t+1} given the reward function $r(S_t, A_t, S_{t+1})$. The agent must learn a policy π that maximizes the *return*—the sum of future rewards:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ldots = \sum_{i=0}^{\infty} R_{t+1+i}. \tag{2.1}$$

Learning against this return can be difficult, thus it is common for the agent to learn to maximize a proxy return instead such as the geometrically discounted return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{i=0}^{\infty} \gamma^i R_{t+1+i}. \tag{2.2}$$

The term $\gamma \in [0, 1)$[1], known by several names including the discount or the timescale, determines how much emphasis is placed on future rewards. If $\gamma = 0$ then the agent is only concerned about immediate rewards and as γ increases more weight is added to future rewards.

In order to determine the best course of action many RL algorithms rely on *value* estimates:

$$v_\pi(s) \doteq \mathbb{E}_\pi \Big[G_t | S_t = s \Big]. \tag{2.3}$$

Value gives the expected return under a given policy and discount starting from a particular state. The state-action value gives the expected return from following policy π after taking action a from state s:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \Big[G_t | S_t = s, A_t = a \Big] \tag{2.4}$$

Policy learning algorithms can then use this information to learn the best course of action to take.

Note, that to simplify notation, we will write drop the policy subscript of the value. It should be assumed that, unless otherwise specified, the value is taken with respect to the agent's behavior policy, π. Further, we will use v to denote the true value function and V to denote an estimate.

[1]Note that γ can be 1 if the trajectory is guaranteed to terminate, as in the episodic case.

2.2 Temporal-Difference Learning

Ideally, real-world agents would be able to learn from each new experience as it happens. We call this *online* learning. When learning value estimates from experience we update the estimates towards samples of the return. However, to do so in an online fashion, the agent cannot wait until the full return has been collected at the end of an episode. Temporal-difference (TD) algorithms solve this by using a technique known as bootstrapping in which value estimates are combined with partial returns to provide a sample of the estimated return. Thus, for a one-step TD update each transition tuple $(S_t, A_t, S_{t+1}, R_{t+1})$ gives us the well-known TD-error (Sutton and Barto, 2018):

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t). \tag{2.5}$$

Assume that the value estimator is parameterized by weights $\mathbf{w}$: $V(s; \mathbf{w})$. These weights can be adjusted so as to minimize the squared TD-error, which is commonly used as the loss function. Various optimization techniques such as stochastic gradient descent are based on taking a small step, controlled by the step-size parameter, $0 < \alpha < 1$, in the direction of the negative gradient of the loss:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} \mathcal{L}_t. \tag{2.6}$$

Note that, in most TD algorithms, a *semi-gradient* approach is used, in which the bootstrapped target return is assumed to be independent of $\mathbf{w}$ when taking the gradient.

In the tabular setting, in which each state is independently and uniquely represented, we can write our weights as $\mathbf{w}_{i;t} = V_{\pi;t}(s_i)$, computing the *semi-gradient* then gives us the update:

$$V_{t+1}(S_t) \leftarrow V_t(S_t) + \alpha \delta_t. \tag{2.7}$$

Bootstrapping introduces bias in the estimates. On the other hand, updating towards the full return (the Monte Carlo return) can have high variance. We can instead use the n-step return to balance this tradeoff. As the name implies, the n-step return uses n steps of the full return before bootstrapping:

$$G_t^n \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_j^{\lambda}(S_{t+n}), \tag{2.8}$$

where V_j is used to indicate the estimate available at the time the return is calculated, which may not be directly tied to t (e.g. training from a replay buffer may disassociate t and j).

A family of algorithms known as TD(λ) averages over many n-step returns by using an alternative return, known as the λ-return:

$$G_t^\lambda \doteq R_{t+1} + \gamma(1-\lambda)V_j^\lambda(S_{t+1}) + \gamma\lambda G_{t+1}^\lambda, \tag{2.9}$$

where $\lambda \in [0, 1]$ trades off bias and variance in the value estimate. For many learning settings better performance can be achieved by setting λ to some value slightly less than 1, e.g. 0.95. However, this equation, which is referred to as the *forward view*, requires the full return, which, again, is not available until episode termination. To implement TD(λ) in an online fashion we use a mechanism known as traces. Traces keep track of the past states the agent has visited (Sutton and Barto, 2018; van Seijen and Sutton, 2014) and are used to push back corrections to the value estimates of previously visited states. In this way, traces assign credit to previously visited states. This is referred to as the *backward view* of the λ-return. TD(λ) with accumulating traces, $\mathbf{z}$ is shown below. Our value estimate, V, is parameterized by $\mathbf{w}$.

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t) \\
\mathbf{z}_t &= \lambda\gamma\mathbf{z}_{t-1} + \nabla_{\mathbf{w}} V_t(S_t) \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t
\end{aligned} \tag{2.10}$$

Another aspect of TD learning is on-policy vs. off-policy learning. In on-policy learning the agent executes a behavior policy and learns a value estimate for that policy. In off-policy learning the agent follows one policy, the behavior policy, b, while learning a value estimate for a different target policy, π. In short, off-policy learning is following one policy while learning about another. This allows an agent to learn about many policies simultaneously. For one-step returns we have a straight forward off-policy learning algorithm, Q-learning, which learns a value estimate with respect to the greedy policy and forms estimated returns as follows:

$$G_t = R_{t+1} + \gamma \max_a Q_\pi(S_{t+1}, a).$$

In general though, off-policy learning is impacted by both the difference in observed returns and the difference in the distribution over states. In practice, correcting for the

distribution over states is generally ignored. To correct for the difference between the probabilities of single-step action selection in estimating V it is common to reweight updates using the *importance sampling* ratio:

$$\rho(a, s) = \frac{\pi(a|s)}{b(a|s)}. \tag{2.11}$$

The importance sampling ratio can introduce high variability in the update targets, making learning difficult and even unstable. Further, off-policy learning using TD(0) with linear function approximation has been shown to diverge. The GTD(λ) algorithm (Maei, 2011) was designed to address the instabilities in off-policy TD learning with linear function approximation:

$$\begin{aligned} \mathbf{z}_t &= \rho_t(\mathbf{z}_{t-1}\gamma\lambda + \phi(S_t)) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha(\delta_t \mathbf{z}_t - \gamma(1-\lambda)\phi(S_{t+1})(\mathbf{z}_t^\top \mathbf{h}_t)) \\ \mathbf{h}_{t+1} &= \mathbf{h}_t + \alpha_h(\delta_t \mathbf{z}_t - (\mathbf{h}_t^\top \phi(S_t))\phi(S_t)), \end{aligned} \tag{2.12}$$

where the secondary weight vector $\mathbf{h}$ is used to correct the estimate of the gradient and α_h is the step-size used to adjust $\mathbf{h}$. Note that we have introduced the concept of function approximation of state $\phi(s)$, which is discussed in the next section. For more detailed discussions on off-policy learning see Maei (2011) and Mahmood (2017). The majority of the work in this dissertation focuses on the on-policy setting, thus, we primarily use TD methods rather than GTD or related off-policy algorithms.

2.3 Representation

Good representations of state are crucial for all AI and machine learning algorithms. In the MDP setting we talk about *state*—the configuration of the world—as a discrete thing that is unique and knowable. If the agent can distinguish each state uniquely from its observations then the agent can use a tabular representation in which all states are represented uniquely and independently as a fixed length vector. All elements of this vector are zero except the state that the agent observes, which is indicated by a one. This is known as a tabular representation. In the limit this will give the most accurate predictions as there is no conflation between states. However, there are several reasons why this is undesirable or even impossible. First, consider again the case where it is possible to use a tabular representation. A tabular representation does not guarantee fast learning. Similar states are treated completely independently and the learning done in one state does not benefit those states that

are similar. That is, tabular representations do not generalize. Further, in reality, for even moderately complex settings, tabular representations are simply not possible; state cannot be uniquely observed or the state space may be incredibly large or even infinite.

To deal with this we use function approximation in which the system takes as input an observation $X(S_t)$, which is a function of the true state of the world, and maps it a feature vector $\phi_t \equiv \phi(X(S_t))$. The value estimate is then given as $V(\phi(s))$. The question then becomes, what is a good representation? Ultimately, ϕ should capture the properties of the environment that identify the similarities and the differences of the underlying state and may even be a summary of the past. Properties that show the similarities of the underlying state allow the agent to learn faster by generalizing. Even in the tabular setting, it can be desirable to use representations that capture the similarities of states, as they enable more sample efficient learning (as an example, consider the symmetries in tic-tac-toe). Properties that discriminate the important differences of the observations ensure accuracy for specific situations.

The next question to ask is, where do good features come from? In many applications, the system designer hand engineers features building in their own understanding of the problem. This is generally a difficult and costly process. In fact, in many fields of machine learning, such as vision and electromyographic control of prosthesis, countless years of research has gone into finding good features. Instead, it is desirable to have the agent construct the representation on its own from its own experience. In recent years one of the major breakthroughs in machine learning has been the ability to learn representations directly from data in an end-to-end fashion using deep neural networks (Goodfellow et al., 2016). An overview of modern representation learning techniques can be found in Bengio, Courville, et al. (2013).

Once we have a representation, ϕ, we can make an estimate, $\hat{y} = g(\phi)$. As a common example, we might simply use linear function approximation (LFA) that simply involves taking an inner product between our feature vector and a learned set of weights:

$$\hat{y} = \phi^\top \mathbf{w},$$

and thus we can write our value estimate as:

$$V(s) = \phi(s)^\top \mathbf{w}.$$

The goal of the learning algorithm is then to learn the set of weights, $\mathbf{w}$, which minimizes the chosen loss.

In the following sections we discuss methods used in this dissertation for constructing ϕ.

2.3.1 Tilecoding

Tilecoding (Sutton and Barto, 2018) is a non-linear method of discretizing real-valued inputs, which is used frequently throughout this dissertation. In tilecoding, an individual input can be tilecoded by itself or multiple inputs can be combined as in the 2D case shown in Figure 2.2. A single tiling is formed by subdividing the input space. The output of the tiling is a one-hot vector where all elements are 0 except for the bin occupied by the input values. A single tiling is limited in its representational power. Instead we use multiple overlapping tilings, each offset from the others. In this way the tilings are able to capture both course generalizations and finer details. Naive tilecoding can quickly produce extremely large feature vectors. To get around this, hashing is often used to limit the size of the feature space. A side-effect of hashing is that it can produce collisions when multiple inputs are mapped to the same hash. If we can assume that the observations of the agent only cover a small portion of the input space, which is generally not an unreasonable assumption, then such collisions can be rare.

2.3.2 Deep Learning

Deep learning (DL) refers to a family of function approximation techniques in which many non-linear function approximators are layered on top of each other with data flowing sequentially through the layers. This is typically done with artificial neural networks (ANN) in which each layer is composed of many artificial neurons, such as perceptrons (Figure 2.3). An artificial neuron takes input x and performs some non-linear operation parameterized by weights w and bias b and produces an output $\hat{y} = g(x; w, b)$. An example operation would be to take an inner product between w and x, add b, and then pass the result through a non-linear activation function like the rectified linear unit (ReLU): $max(0, w^\top x + b)$. Without a non-linearity the network could not represent non-linear functions. Deep networks stack many layers of non-linear function approximators one after the other. The depth of these networks is key to their compact representational power and tractable training times (Bengio and Delalleau, 2011).

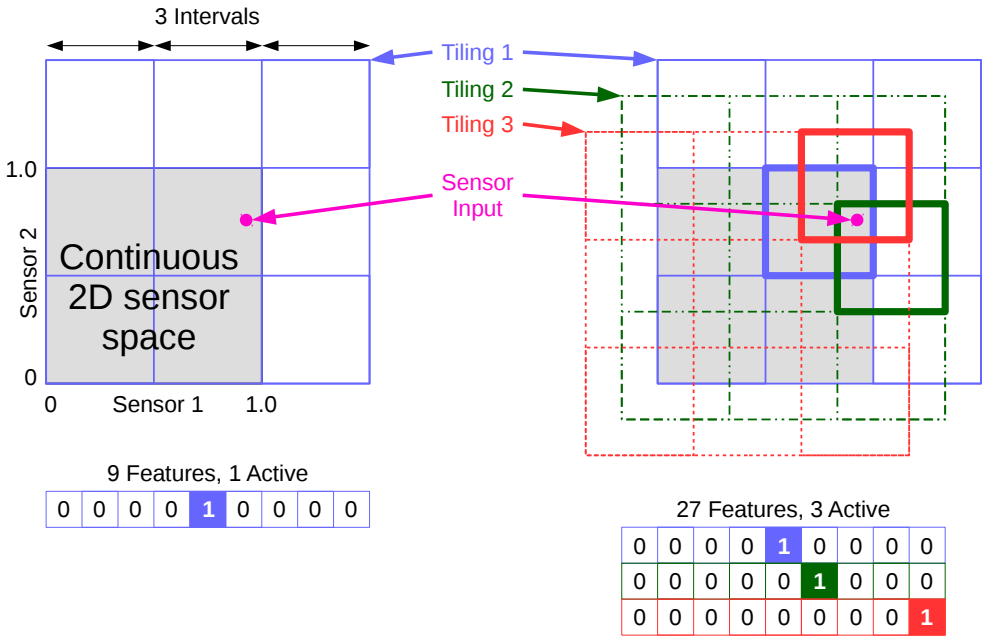


Figure 2.2: **Tilecoding.** In this example a 2D sensor space is normalized and discretized into bins with width 0.5. Each tiling then has 9 possible bins, although in the case where the bins are perfectly aligned, as on the left, only 4 would be used. When multiple tilings are combined together each is shifted allowing the discretization to capture both coarse generalizations and fine refinements. (Figure taken from Sherstan (2015))

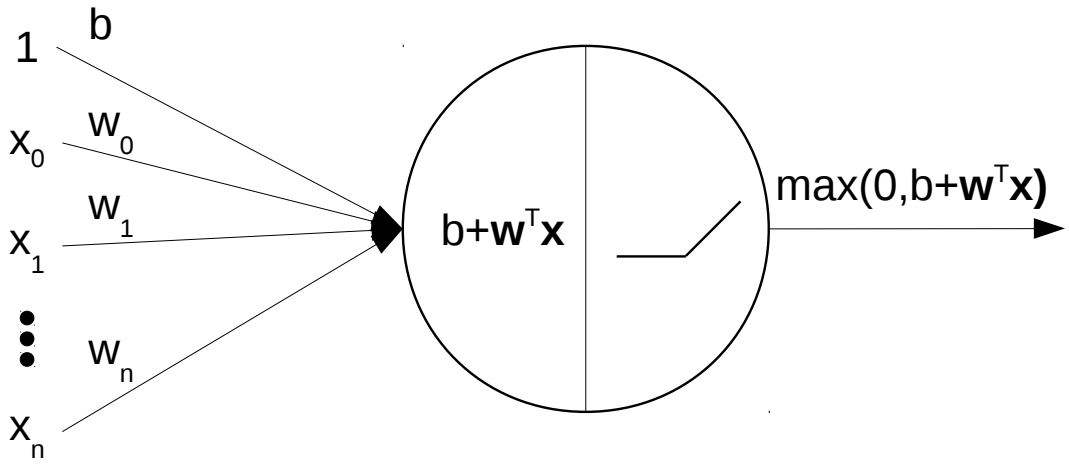


Figure 2.3: Perceptron. An inner product followed by a non-linearity, shown here with a ReLU activation.

The most common approach to training an ANN is the use of backpropagation (Goodfellow et al., 2016). Backpropagation is applied as follows. First a loss, $\mathcal{L}$, is computed at some

head of the network. For each parameter, w_i, in the network gradients can then be computed $\frac{\partial \mathcal{L}}{\partial w_i}$ and weights updated by gradient descent. The key to backpropagation is the observation that the gradient can be computed through the entire network by the use of the chain rule. Modern computational libraries such as TensorFlow (Abadi et al., 2015), PyTorch (Paszke et al., 2019), and MXNet (Chen et al., 2015) handle this process automatically.

2.3.3 Deep Reinforcement Learning

Mnih, Kavukcuoglu, et al. (2015) demonstrated the successful combination of deep learning with RL with *Deep Q-Networks* (DQN). DQN was used to train agents to play Atari games directly from pixel input using end-to-end training driven by reward. To process images DQN uses a convolutional neural network (CNN) followed by a fully-connected layer. Following the representation network is a Q-value network that consists of $|\mathcal{A}|$ nodes. In order to capture history, multiple consecutive image frames are stacked together before being passed into the network. The network is not trained from each sample as it comes in, rather, transition tuples are saved to an *experience replay buffer*. For each training step, the agent samples a minibatch of transitions uniformly from the replay buffer and computes the one-step TD-error for each transition. The TD-error is bootstrapped from the greedy action choice of the next state's estimated Q-values. DQN is used in Section 8.2. In Chapter 7 we use a variant of DQN called Rainbow. The implementation of Rainbow that we used makes three changes to DQN. The first is to use prioritized sampling from the replay buffer where samples are weighted according to the magnitude of their past training errors. In other words, a sample that has had high error has a higher probability of being sampled. The second change is to use n-step returns instead of one-step (Equation 2.8). The final change is to estimate the distribution over returns rather than just the mean (Bellemare, Dabney, and Munos, 2017). All these changes have been found to improve performance over DQN. Since Mnih, Kavukcuoglu, et al. (2015), the study of Deep RL, as it's referred to, has taken off. Not only is it being studied in earnest in research settings, but it is being used to solve practical problems in the real-world as well.

2.4 General Value Functions

General value functions (GVFs) make two relaxations to the value function definition we have already considered (Sutton, Modayil, et al., 2011). First, we are free to choose any signal available to the agent as the prediction target, not just reward. We refer to the prediction target as the *cumulant*, C. Secondly, the discount parameter, γ, is replaced by a transition dependent continuation function: $\gamma_{t+1} \equiv \gamma(S_t, A_t, S_{t+1})$ (White, 2017) (Note that given this definition γ need not lie in $[0, 1]$, and can even be complex valued (De Asis et al., 2018)). This function is referred to by several names in the literature including the continuation function, discount and timescale. With these two generalizations we define the return as

$$\begin{aligned} G_t &= C_{t+1} + \gamma_{t+1} C_{t+2} + \gamma_{t+1}\gamma_{t+2} C_{t+3} + \dots \\ &= \sum_{i=0}^{\infty} (\prod_{j=1}^{i} \gamma_{t+j}) C_{t+1+i}. \end{aligned}$$

Like a value function, a GVF is defined by three components: the policy, the timescale, and the prediction target. GVFs allow the agent to express representation elements in the form of predictive questions. Consider the following examples for a mobile robot:

- "If I drive forward, how much current will my motors draw over the next 10 timesteps?"

  Policy: Drive forward
  Cumulant: Motor current
  Timescale: 10 ts ($\gamma = 0.9$)

- "If I drive forward, how long until my bump sensor is activated?"

  Policy: Drive forward
  Cumulant: 1
  Timescale: $\begin{cases} \text{A large value, say 0.99} & \text{Bump sensor} = 0 \\ 0 & \text{Bump sensor} = 1 \end{cases}$

The term $(1 - \gamma)$ can be treated as the probability of terminating on the next step, therefore γ can be viewed as the probability of continuation. We can then define the expected number of timesteps till termination for a fixed γ as (see Sherstan (2015) for a derivation):

$$\tau = \frac{1}{1 - \gamma}. \tag{2.13}$$

The variable τ has units of timesteps, ts. We will often refer to the timescale of a prediction in an imprecise way, say as 10 ts. Unless otherwise stated this refers to the value

of τ and has a corresponding value of γ. Timescale will be discussed in more detail in Chapter 7.

GVFs have the same definition as value functions, thus, we are free to use any of our usual learning methods, such as TD(λ).

2.4.1 PSRs and TD-nets

Predictive state representations (PSRs) (Littman et al., 2002) and temporal-difference networks (TD-nets) (Sutton and Tanner, 2005) are predictive approaches to representation, which precede GVFs. PSRs are motivated by the problem of partially observable Markov decision processes (POMDPs). In POMDPs the agent knows the possible states but is unable to directly observe them. Instead the agent constructs *belief states*—distributions over latent states. PSRs represent state as a collection of *core tests*. A test gives the probability of observing a specific sequence of observations, $\mathbb{o} = [o_{t+1}, o_{t+2}, \ldots, o_{t+T}]$, starting from time t with history h_t and following a specific sequence of open-loop actions, $\nu = [a_{t+1}, a_{t+2}, \ldots, a_T]$. Thus, the test output is $Pr(O_{t+1}, O_{t+2}, \ldots, O_{t+T} = \mathbb{o}|h_t, \nu)$. Given a sufficient set of tests that are linearly independent, each state in the POMDP can be uniquely described. This set is referred to as the *core* tests. Finding this set can be very difficult. Further, the idea that all possible states are known or can be known does not hold in the real world. Despite these difficulties, the predictive syntax of PSR tests may still be useful. However, GVFs provide a more general syntax, which may be able to describe PSR tests as well. A key difference between GVFs and PSRs is the policies over which they can be defined. PSR tests are defined with respect to a fixed sequence of future actions, which do not depend on environmental response (open-loop). In contrast GVFs are defined over general, state-dependent policies (closed-loop), which can be stochastic and of variable length. This property allows GVFs to generalize better than PSRs (Schaul and Ring, 2013).

TD-nets and GVFs are more general than PSRs, allowing the prediction of arbitrary scalar signals. GVFs can be seen as a natural generalization or relaxation of TD-nets with options (Sutton, Rafols, et al., 2006). The semantics are the same, but GVFs do not necessarily impose a particular network structure or recursive state representation, as was done with TD-nets. Thus, it is more accurate to say that TD-nets are one particular instantiation of a GVF-based predictive architecture.

2.4.2 GVFs So Far

GVFs, in their current form, were first introduced by Sutton, Modayil, et al. (2011), in what they referred to as the *Horde* architecture. In this paper they showed that it was possible to learn and make multiple GVF predictions simultaneously on a robot in parallel using *off-policy learning*. Modayil, White, et al. (2014) extended this work to demonstrate that thousands of GVF estimates, over diverse sensory spaces, could be practically learned in an online fashion using the sensorimotor data of a simple mobile robot. This illustrated a concept known as *nexting*—continuously making numerous short timescale predictions with respect to the agent's local situation, something believed to be done by humans and animals (Gilbert, 2006).

One of the first ways in which GVFs were used was with *Pavlovian control*—reflexive responses to predictions of external stimuli. Modayil and Sutton (2014) demonstrated that GVFs could be used to trigger fixed reflexive policies whenever a particular prediction exceeded a threshold. In one experiment their mobile robot learned to predict wheel stall events, a dangerous situation where the robot might continuously push against a wall, potentially damaging the motors from overheating. Once the robot's prediction for the stall event exceeded a threshold a command was sent to turn off the motors. In a second experiment they demonstrated how GVF predictions could be used to reduce communication lag by preemptively taking action based on predicted commands. GVF-based Pavlovian control has also been demonstrated in research aimed at restoring walking for paralyzed humans. In the work of Dalrymple et al. (2020), electrical stimulation of an anesthetized animal's spinal cord was driven by a state-machine based gait controller to produce walking. GVFs were used to predict various sensorimotor signals. When predictions exceeded specified thresholds the state machine was transitioned to the next state in the gait cycle. To date, GVFs with Pavlovian control may still represent one of the most effective ways that GVFs can be deployed in current robotic systems.

One of the goals of the Pilarski lab and the *Bionic Limbs for Improved Natural Control (BLINC)* lab, of which I have been a member, is to create intelligent prosthetic limbs that can predict their user's needs and intentions and assist the user, filling in the gaps in control. To this end, we have focused much of our research on the use of GVFs in prosthetic control. A common method for controlling powered prosthetic arms is the use of *electromyography*—the

electrical signals produced by muscle contraction. Sensors are placed on a prosthesis user's residual limb over the muscle belly and the user can then send voluntary control signals in the form of muscle contractions. However, with such a system it is common that the user is only able to control a single joint at a time and must provide some additional signal to switch which joint they are presently controlling, a tedious process. Edwards et al. (2016) used GVFs to predict which joint the user would select next. Then, when the user would switch joints, rather than giving the joint list in a fixed order, as is typical, the joint list would be reordered by most probable as predicted by the GVFs. This enabled the user to select the desired joint with fewer control signals and complete their task quicker. Additionally, GVFs have been used in several prediction and movement control experiments with a prosthetic arm. In the work of Pilarski, Dick, et al. (2013) a user controlled elbow and hand joints while the computer agent controlled the wrist. The agent learned to predict wrist joint position as a function of the current arm configuration. These predictions were then used in two different control scenarios: 1) to directly drive the wrist joint via Pavlovian control and 2) as part of the state information used by an actor-critic policy learner. Sherstan, Modayil, et al. (2015) created a system that assisted a user in controlling multiple joints simultaneously. In our system the agent first observed the user as they controlled a prosthetic arm one joint at a time, learning GVF predictions of future joint positions. The predicted joint positions were then mapped directly to control the joints that were not presently being controlled by the user. That is, if the user controlled the elbow joint, the agent would use its prediction of future shoulder joint positions as movement targets for the shoulder joint. Vasan and Pilarski (2018) showed that accurate GVF predictions about target joint positions could be learned from visual information. Parker et al. (2019) used GVFs to predict joint motor loading in order to provide users with predictive feedback when controlling a robot arm. This allowed the user to more effectively avoid overloading the joints and to complete their task quicker.

There are several general ways in which GVFs might be used. The first is as a predictive state representation. An RL agent learns a mapping from its state representation to action (Pavlovian control uses a fixed mapping from state to action). When an agent does not have a complete observation of the world-state, predictive representations provide a powerful way of disambiguating between observations. Further, predictive representations of state allow an agent to generalize its learning to configurations not seen before (Littman et al., 2002; Rafols et al., 2005; Schaul and Ring, 2013). GVFs have been shown to generalize better than an

alternative predictive representation known as PSRs (Schaul and Ring, 2013). Using GVFs as predictive state representation is the primary usage by which I motivate my research. A second way in which GVFs might be used is to drive representation learning as auxiliary tasks as in the UNREAL architecture (Jaderberg et al., 2017). This approach is discussed further in Chapter 8. Another way in which GVFs can be used is to form policies themselves. For example, in Sutton, Modayil, et al. (2011) a mobile robot followed a random behavior policy while learning a GVF that maximized the intensity of light in one of its sensors. The robot then followed the GVF policy by selecting greedy actions that maximized the estimated light intensity. The result was that the robot learned to seek light. In work by Riedmiller et al. (2018) they build off these ideas. In their work numerous GVFs are learned that maximize different, hand selected, signals that are grounded in the agent's sensorimotor stream. The agent then learned a policy that selected which GVF policy to follow as a function of its current state. This resulted in improved performance in learning complex robot tasks with sparse reward. Finally, taken together, GVFs form a model, as we have already discussed. One way in which models are commonly used in AI is for planning courses of action. Planning with GVFs is an open, and largely unexplored, avenue of research that I do not address in this dissertation.

2.5 Reuse

In reinforcement learning, agents act in the world to solve tasks. Most RL, but not all, research has focused on solving single tasks. In this setting a model helps the agent to understand the world and predict the outcomes of its actions, thus enabling better decision making. Models are even more important in the settings of *lifelong learning* (Thrun and Mitchell, 1993), in which an agent is given multiple consecutive tasks, and the related setting of *continual learning* in which an agent undergoes "the constant development of complex behaviors with no final end in mind" (Ring, 1994). In these settings the agent is expected to continually learn new skills. This is facilitated by leveraging existing skills and models. Models enable an agent to reuse its past experience in new settings to accomplish new tasks. Ultimately, this incremental process sets up an ever expanding loop of capability and world understanding — current skills enable new experiences that are used to build models, models enable more complex and refined skills, which in turn enables new experiences and leads to

more complex and refined models.

However, the structure with which models are represented affects the degree to which they can be reused. The following subsections describe two approaches to representations that may enhance sample efficiency and reuse by enabling an agent to generalize its model across different tasks and settings.

2.5.1 The Successor Representation

The Successor Representation (SR) was introduced by Dayan (1993) as a predictive representation for policy learning. It separates value estimation into two components, one for the dynamics of the environment and the other for the reward. If we write the tabular Bellman equation in vector format and solve for value, $\mathbf{v}$, we have:

$$\begin{aligned} \mathbf{v}_\pi &= \bar{\mathbf{r}}_\pi + \gamma P_\pi \mathbf{v}_\pi \\ (I - \gamma P_\pi)\mathbf{v}_\pi &= \bar{\mathbf{r}}_\pi \\ \mathbf{v}_\pi &= (I - \gamma P_\pi)^{-1}\bar{\mathbf{r}}_\pi \end{aligned}$$

where:

$\bar{\mathbf{r}}_\pi$ - the average one-step reward from each state: $\bar{\mathbf{r}}_\pi[i] = \mathbb{E}[R_{t+1}|S_t = s_i]$

P_π - the transition matrix: $P_{ij} = Pr_\pi(S_{t+1} = s_j|S_t = s_i)$,
the probability of transitioning from state s_i to s_j
when following policy π

I - the identity matrix

The SR is defined as the first factor in this factorization:

$$\mathbf{\Psi}_\pi = \sum_{t=0}^{\infty}(\gamma P_\pi)^t = (I - \gamma P_\pi)^{-1}. \tag{2.14}$$

The SR captures the dynamics of the environment with respect to a policy, π and discount γ. Given a policy, this factor is completely independent of the reward. The second factor, $\bar{\mathbf{r}}_\pi$, is the one-step expected reward from each state under policy π. In the function approximation setting we can write the SR as (this is referred to as *Successor Features* in other literature,

such as (Barreto, Borsa, et al., 2018; Kulkarni et al., 2016)):

$$\begin{aligned}\boldsymbol{\Psi}_\pi(s) &= \mathbb{E}_\pi[\phi(S_0) + \gamma\phi(S_1) + \gamma^2\phi(S_2) + \ldots | S_0 = s] \\ &= \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t \phi(S_t) | S_0 = s\Big].\end{aligned}$$

A TD-error can then be derived for the SR and it can be learned using standard TD learning methods:

$$\boldsymbol{\delta}_{\boldsymbol{\Psi};t} = \phi(S_t) + \gamma_{t+1}\psi\big(\phi(S_{t+1})\big) - \psi\big(\phi(S_t)\big).$$

If we consider the equation $V(s) = \boldsymbol{\Psi}(s)^\top \bar{\mathbf{r}}$ then we see that this is a form of linear function approximation in which the SR forms the input feature representation and $\bar{\mathbf{r}}$ is the weight vector that must be learned.

The SR can be seen as a temporally extended model, capturing how the environment changes with response to action. Like typical model-based RL algorithms (in which the agent learns a one-step transition and reward model) the SR can be used for *planning*, where the agent updates its value estimates and policies by taking imagined actions using its model (Russek et al., 2017). In fact, it has been proposed that something like the SR is used in the hippocampus of mammals to enable predictive localization and planning in space (Stachenfeld et al., 2017). One of the current research focuses with the SR is in the field of *transfer learning* (Barreto, Borsa, et al., 2018; Kulkarni et al., 2016; Ma et al., 2018; Zhu et al., 2017; Zhang et al., 2017; Lehnert et al., 2017). Loosely, this is defined as learning something in one context and being able to reuse it in another. Because the SR captures the dynamics of the environment under a policy independently of reward it can be reused to evaluate any signal function, including reward functions, with respect to the same policy, enabling transfer across many signal functions.

Clarification About Indexing with the SR

I wish to clarify an issue related to how the cumulant is indexed in the SR as it has caused myself and colleagues considerable confusion. The SR is typically said to predict how often a state will be visited in the future. Although conceptually intuitive, this is not technically accurate and can lead to the wrong intuition regarding the learning updates. The usual thing

to do in TD is to count the cumulant as the observed signal when transitioning from S_t to S_{t+1}. If we are counting future states then we would want to use a cumulant of $\phi(S_{t+1})$. While this is certainly something that can be predicted, it does not give the SR; the cumulant of the SR is $\phi(S_t)$. I find it best to focus on the definition of the SR as given by Equation 2.14 rather than thinking of it as predicting future state visitations.

2.5.2 Universal Value Function Approximators

Universal Value Function Approximators (UVFAs) (Schaul, Horgan, et al., 2015) are a method for generalizing value estimation across goals and their corresponding policies. The main idea behind UVFAs is to provide a representation of the agent's current goal, referred to as the embedding, as input to its function approximation network. That is, they make estimates a function of both state, s, and goal, g: $V(s, g)$.

For GVF architectures we want to make large numbers of predictions. However, each prediction is policy dependent and there are potentially infinite policies an agent could follow. If predictors are represented uniquely then there are two problems that must be addressed: 1) selecting the policies to make predictions about 2) learning about all the desired policies. By treating the policy as an input to the function approximator it is able to learn generalizations across related policies. This is potentially a very powerful mechanism for enabling compact, tractable, value representations that are sample efficient.

UVFAs perform a complementary function to off-policy learning that was demonstrated in "Unicorn: Continual Learning with a Universal, Off-policy Agent" by Mankowitz et al. (2018). This paper looked at how the UVFA and off-policy learning could be used together in a continual learning setting. The experimental setting was an object collection task set in the 3D gaming environment DeepMind Lab (Beattie et al., 2016). The tasks were defined to be sequential in nature; the key must be collected to open the lock before the door can be opened and then the chest opened. The ultimate goal was to open the chest. Many agents ran in parallel following policies to collect the various items in the environment and feeding their experience back to a central learner, akin to the IMPALA architecture (Espeholt et al., 2018). Q-value estimates were represented using a UVFA. Training was applied in the following way. At the start of each episode the agent selected a target item to acquire, let us call that goal g_i, and followed the corresponding policy that was encoded as Q-values by the

UVFA. When the episode terminated all goals were trained, not just the behavior goal. An update target was formed as an n-step return, where n is the number of steps at which point the agent selected a different action than would have been taken under the target policy corresponding to goal g_j:

$$G_{t,g_j} = \sum_{k=1}^{n} \gamma^{k-1} R_{t+k;g_j} + \gamma^n \max_a Q(a, S_{t+n}, g_j).$$

Here $R_{t;g_j}$ is the reward received at time t with respect to the target goal's reward function. Bootstrapping is done using the UVFA with the embedding corresponding to goal g_j. The researchers found that training in this way enabled better performance than training an agent explicitly on the final goal of opening the chest.

Universal Successor Representation Approximator (USRA) (Ma et al., 2018) combines the UVFA and the SR representation. The SR, the average one-step reward estimator and the policy are all dependent on the goal: $\psi(\phi(s), g), \bar{\mathbf{r}}(g), \pi(\phi(s), g)$. The current environment image is given as the input, s, and the goal is presented as the desired environment image. This is an interesting idea in that it is able to combine different reuse approaches.

The idea of treating estimator parameters as part of the input to the function approximator network is one that I build on further in Chapter 7 to enable generalization across the prediction timescale parameter γ.

Part I

Questions: Representation with GVFs

Part I looks at the types of predictive questions that can be encoded as GVFs. Chapter 3 introduces the motivating ideas for the rest of this part of the dissertation. Specifically, we propose that the agent should use internally generated signals, introspective measures, as part of its state representation. Further, predicting such introspective signals with GVFs may provide additional information to the agent. In Chapter 4 we provide empirical investigations of the behavior of several introspective measures, examining how they may encode information about the agent's interaction with its environment. In Chapter 5 we look at one particular introspective measure, the variance of the return, and present a new algorithm for estimating it using a series of GVFs. This part of the dissertation represents a fertile opportunity for further research.

Chapter 3

Introspective Agents

The core tenet of this chapter is the idea that introspective measures—measures about an agent's own state and learning process—should be included as part of the agent's state representation. While there are numerous works on curiosity, exploration, and intrinsic motivation that are clearly related, we believe this to be an understudied area of research. We further advocate that such introspective measures serve as prediction targets for GVFs and that some introspective measures could be captured as GVFs themselves. This chapter first outlines our position in detail before relating it to existing research. In the third part of the chapter we describe various introspective certainty measures and suggest how they might be useful. Introspective measures would augment an agent's state representation with information that is currently unavailable in standard approaches. We posit that the inclusion of such information would be particularly useful in continual learning agents faced with a complex, dynamic world.

3.1 Introspection and Prediction

In RL we commonly describe an agent's interaction with its environment as a Markov Decision Process, which models the dynamics of the environment as transitions from one discrete state to the next. In complex scenarios this concept of *state* does not hold, or at least cannot be explicitly observed or described. Instead, an agent must rely on its own representation of state, an approximation that we will refer as the *agent's state*. Ideally, an agent's state should provide a sufficient, minimal and well-factored description of the current configuration of the world. By sufficient we mean that the representation contains all the information that the agent needs in order to make optimal decisions. By minimal we mean that only

information that is needed is presented; there is nothing extra to confuse the agent. By well-factored we mean that all the factors of variation are well separated. In the continual learning setting an agent works to construct and improve this state representation over time.

Commonly, an agent's state representation is dependent only on external sensorimotor information. However, this misses part of the picture—the agent's own internal configuration, its internal state. Here we suggest that *introspective signals* should be included in the agent's state representation. These are signals relating to the agent's own learning mechanisms—for example, its prediction errors, weight changes, and other time-varying meta-parameters. Such signals are generally treated as just part of the algorithmic learning system and ignored as part of state. There are also a range of signals that quantify the agent's interactions with its sensorimotor stream—for example, statistics about state or feature-space visitation and statistical properties of sensorimotor signals themselves. Thus, our first conjecture is as follows:

1. An agent's state representation will be improved by including introspective signals.

GVF research has focused on predictions about signals external to the agent—signals in the agent's sensorimotor stream. However, the introspective signals previously mentioned may also serve as prediction targets for GVFs. Thus, our second conjecture is:

2. An agent's state representation will be further improved by including predictions of introspective signals.

Ultimately, the quality of an agent's state affects its ability to make good decisions. We suggest that including introspective signals as part of the agent's state should further improve its decision-making capabilities.

The terms *certainty* and *confidence* are used in many different ways without a universal consensus. Pouget et al. (2016) provide a perspective for thinking about these words from a neurological perspective of animal and human decision making. For the purposes of this chapter and the next we will use their definitions. Given some predicted variable, μ, they define *certainty* as the width of the distribution of possible outcomes given an input source. They propose that each measure of certainty is taken with respect to some context

or modality. For example, in their work they suggest that different modalities, such as vision (V) and auditory (A) data, would each have their own distribution for prediction of μ: $p(\mu|V), p(\mu|A)$. Mathematically, certainty might be measured as the inverse entropy or inverse variance of those distributions. However, neither entropy or variance discriminate the sources of uncertainty, whether they be from insufficient data, an inaccurate model, or intrinsic stochasticity in the system. *Confidence* is defined as the probability that an agent's choice is the correct one, given all of its information. Confidence is defined in terms of an agent's own estimation, rather than the ground truth. This idea of confidence is relative to an agent's own policy and may change depending on the situation. For the RL setting we can define confidence as the probability that the agent's action selection is the optimal one: $p_\theta(\pi_{\mathbf{w}}(\phi) = \pi^*(\phi)|\phi \equiv \phi(V, A))$, where we have defined: $\pi_{\mathbf{w}}$ as the agent's policy parameterized by $\mathbf{w}$, π^* as the optimal policy, ϕ as the representation of the input and we denote p as being parameterized by θ to make it clear that confidence is subjective, rather than based on the ground truth; it is always an estimate.

While these mathematical definitions suggest explicit representation of distributions, certainty may not be explicitly known or modelled. Instead, we suggest that various introspective measures might serve as substitutes, fulfilling the same purpose of informing an agent's confidence. We will refer to these as *certainty measures*. Thus, our third conjecture follows:

3. Certainty measures, a class of introspective measures, inform an agent's estimate of its confidence.

These ideas can be applied to GVFs. An estimated answer to a GVF question simply provides a prediction—a scalar value—but does not provide any sense of the certainty of that prediction. This leads to our fourth conjecture.

4. Certainty measures allow an agent to determine when and how to use a prediction.

There are many different measures that describe a distribution and might be used to characterize its uncertainty. Examples include variance and entropy as well as higher moments like skew and kurtosis. However, these measures are a summary of all the factors of uncertainty. Different types of introspective certainty measures, such as those we will describe in Section 3.3.2, may provide different views into the data, the environment, and the

learning process. This may allow an agent to identify and correct the sources of uncertainty. Thus, we make our final conjecture.

5. An agent's decision-making and learning processes will be improved by including multiple certainty measures, not just one.

These conjectures provide the motivation for the remainder of the work presented in Part I.

3.2 Related Works

The study of curiosity and intrinsic motivation—self-driven exploration—is filled with various introspective measures. Oudeyer and Kaplan (2009) provides a summary of approaches to intrinsic motivation. In this study they subdivide the approaches into four different categories: 1) information theoretic and distributional, 2) predictive models, 3) competence-based, and 4) morphological models. In these settings the introspective measures are all used to produce a motivating reward that is given in addition to the agent's main task reward. The information theoretic approaches construct probability distributions over states and transitions. From these distributions they derive numerous approaches. There are those that reward agents for visiting unlikely states, or those that reward the agent for reducing uncertainty or entropy. There are approaches that reward the agent for surprise—violations of strongly held beliefs, and there are those that reward for familiarity. Approaches based on predictive models provide some of the same ideas, but ground their intrinsic motivation directly to prediction errors. The information theoretic methods might be seen as the Bayesian approach, while the predictive model methods might be seen as frequentist. Competence-based approaches construct measures based on the agent's improvement at achieving self-determined goals. Different measures are constructed that encourage an agent to focus on different types of goals. In some approaches the agent is motivated to work on goals for which it presently performs poorest, in others it is encouraged to maximize its competency and yet other approaches reward the agent for maximizing the amount of progress it can make in learning. The morphological category considers statistical properties of the sensorimotor stream itself. The agent is rewarded for achieving some statistical goal in the sensorimotor space such as maximizing the signal correlation across as many signals as possible, or minimizing or maximizing the variance across the signals.

Introspective measures are heavily used in *exploration*—how an agent chooses its actions and goals so as to sample the state-action-reward space. RL agents face a continual trade-off of whether to stick to what is known, exploitation, or try something new, exploration; to achieve optimal performance, maximizing the sum of rewards, an agent cannot strictly do one or the other. This is referred to as the exploration-exploitation dilemma (Sutton and Barto, 2018). Exploration can be beneficial or harmful depending on the circumstances. If the agent expects the environment to be safe or is optimistic then exploration is the right choice. On the other hand if the agent perceives the world, or at least the current situation, as dangerous then exploration may need to be avoided. The value of exploration also depends on the evaluation setting being considered. If we are only concerned with the agent's final performance then exploration should be sought aggressively until sufficient coverage has been achieved. In such settings there is no true consequence for failure along the way, i.e. if the agent dies it just restarts a new episode with no permanent consequence. A key focus, then, is on how to direct the agent's exploratory behavior to achieve sufficient coverage. The works of Bellemare, Srinivasan, et al. (2016) and Machado, Bellemare, and Bowling (2020) give the agent bonuses for visiting states with low visitation count. In the work of Schmidhuber (1991) the agent learns to predict changes in its uncertainty. It then provides a reward to the agent for taking actions that reduce the uncertainty. Ngo et al. (2013) also seek to reduce uncertainty but rather than providing a reward they apply their uncertainty estimate directly to the agent's decision making process. In their system they learn a transition model of the world. For a given state the agent will take actions whose outcome is uncertain. If all actions in the current state are considered certain then the agent will instead plan how to reach a state for which it has uncertainty. White and White (2010) estimate confidence intervals of predicted value that is used to select the action with the highest upper confidence bound. White (2015) used violation of established GVF predictions, a measure of surprise, to switch between different policies. In the work of Mihalkova and Mooney (2006) an agent can request relocation to a different part of the state-space. It will do this if the agent's boredom measure is high or if it predicts that the agent is in a bad situation from which there is no recovery. It will then request to be relocated to a state for which it has high uncertainty about which action is best. All of these methods have encouraged the agent to take actions in pursuit of uncertainty, either through a bonus reward or a heuristic approach to exploration.

On the other hand, if an agent is risk-averse or if there are permanent consequences to bad decisions then the agent might wish to avoid low confidence situations. Such is the case for a continual learning agent. Gehring and Precup (2013) encourages the agent, via a reward bonus, to seek out states that are *controllable*—states for which the expected magnitude of the TD-error is low. Kahn et al. (2017) uses an ensemble method to estimate the uncertainty over projected model rollouts. The uncertainty is then scaled by the movement speed of the agent and given as a penalty. In this way the agent is directly encouraged to take cautious, low-speed, actions in the face of uncertainty. Deisenroth and Rasmussen (2011) achieved extremely fast and robust learning rates by representing the transition model as a Gaussian process. In their setting uncertainty was also used to increase the cost function of a modelled rollout. (Tamar, Castro, et al., 2016) used an estimate of the variance of the return to minimize risk while maximizing reward in financial decisions (see also Sato et al. (2001)).

Introspective measures can also be used to tune model parameters and drive improvements in representation and the predictive models themselves. Ring (1997) used measures of predictive error and weight changes to guide increases in representational capacity for the agent's model. White and White (2010) used confidence intervals to actively adapt the λ parameter, used in credit assignment, in TD(λ). Whereas in White and White (2016) λ is adapted by using the variance of the return.

Introspective measures can be used in conjunction with hand-coded procedures. Daftry et al. (2016) used predictions of collisions with a vision system to improve behavior of a flying drone. During training their system was given numerous trajectories and had to predict whether or not those trajectories would result in a collision. The introspective measure learned was, based on the current camera image, what percentage of projected trajectories would collision be predicted incorrectly. That is, it predicted whether or not the system was likely to get its predictions wrong. If the prediction exceeded a threshold then a safety procedure was initiated.

There are numerous approaches to estimating uncertainty. Bayesian methods (Hu and Kantor, 2017), such as Gaussian processes (Deisenroth and Rasmussen, 2011), are naturally suited to giving uncertainty methods. Other approaches include ensemble methods (Kahn et al., 2017), confidence intervals (White and White, 2010), or directly estimating the distribution (Bellemare, Dabney, and Munos, 2017). These approaches are the more statistically grounded ones. The others listed here tend to use some heuristically defined measure as an

indirect substitute. For example, low visitation count (Bellemare, Srinivasan, et al., 2016; Machado, Bellemare, and Bowling, 2020) is taken as an indirect measure of uncertainty.

All of the approaches listed here apply their introspective measures either by augmenting the reward or cost function or by heuristically modifying the policy directly. Thus far, we have not found any works that directly use their introspective measures as part of the agent's state representation. This is information about the environment and the learning process itself that is often available, but withheld from the agent. Consider the work described in Machado, Bellemare, and Bowling (2020); the agent forms a state visitation count that is encoded in the successor representation. If the agent takes an action leading to a state with a low visitation count then it receives a reward. Notice that all of the information used to compute this reward is already encoded in the successor representation. However, the reward is given after the agent has already taken the action. Thus, the agent does not really learn to directly explore, it does not directly learn to seek states that have a low visitation count, instead it learns to return to states for which past visitation count was low. If, instead, the agent's state representation included the estimates of the successor representation it might be able to learn a policy that seeks out novel states explicitly.

What we see from the numerous different approaches being taken in this field is that there is no single approach and no single measure that captures everything. Instead, an agent needs many measures and to learn multiple generalized approaches to exploration. By including these measures as part of the agent's state information what we expect is that a continual learning agent could learn general strategies of exploration and caution that it can apply to new scenarios. For example, a robot might learn that when it enters a new room that it should reduce its speed until it has built up enough confidence for the new environment (Kahn et al., 2017). An agent might learn when it is safe to explore and when it is not.

The idea of constructing introspective measures is not new. What is new is the suggestion that they should be included as part of the state representation. This is a natural interpretation of the RL framework (See Figure 2.1). The boundary between agent and environment need not occur strictly at the physical boundary between robot and world. Rather, the environment includes anything outside a given RL loop. Thus, we can instead consider that an agent's environment include various computational processes defined internal to the robot itself (Barto, 2013). Further, while some of the methods described in this section can

be seen as making predictions of internally generated signals, such as error (e.g. Gehring and Precup (2013)), ideas connecting introspective measures with GVFs and the need for certainty measures for GVFs is largely unconsidered (see White (2015) for early ideas on this subject).

3.3 Certainty Measures

3.3.1 Recency and State

Certainty measures can be described across two separate axis: recency, and state (See Figure 3.1). *Recency* asks the question, "Have my predictions been accurate lately?" The varying parameter here is how far back in time we consider with extremes being immediate (no history) and all-of-time. Recency measures make the assumption that the underlying state transitions are related in time. That is, a subsequent observation is strongly related to the preceding one, i.e., there are no teleportations of state.

State asks the question, "How accurate have my predictions been when in this state?" State varies by coarseness between a stateless representation (a single bias unit that is always active) and a rich one that disentangles all the factors of variation (a tabular representation in the limit). On their own, recency based measures might be effective for decision making. However, state-based measures offer a number of advantages. The first is that the previously stated assumption, that subsequent observations are highly related, need not hold. A non-linear function approximation across states allows for discontinuities or sharp jumps in the predictions. Secondly, using a stateful measure enables the agent to predict states of low confidence ahead of time and even to plan accordingly. Imagine that the agent moves into new state territory, i.e., states that have not yet been experienced. With stateless recency our confidence would not drop until after our predictions have made errors. With a state-based approach we are able to identify that we are moving into new territory and lower our confidence in anticipation. Finally, state-based measures allow us to provide certainty measures independently of current observations, making them suitable for planning.

3.3.2 Candidate Certainty Measures

In this section we highlight several measures of certainty, which might be used on their own or together. These measures are not necessarily new, many of them have been used

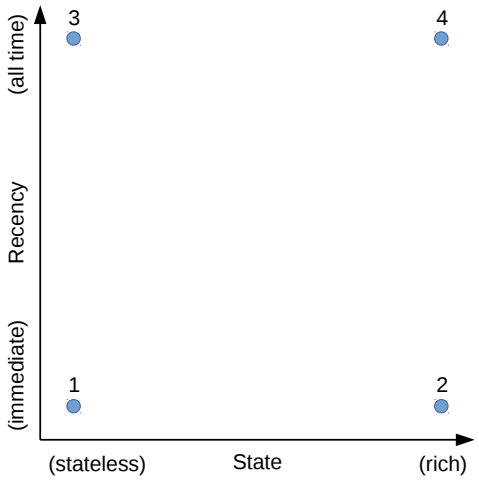


Figure 3.1: Recency and State. Extremes are marked. 1) A measure that captures an immediate statistic, e.g. squared TD-error. 2) A measure that captures an immediate statistic as a function of state, e.g. expected squared TD-error as a function of the current observation. 3) A stateless measure capturing a statistic over all time, e.g. the max observed squared TD-error. 4) A stateful statistic over all time, e.g. the max squared TD-error observed from the current observation.

in different works such as those described in the previous section. Each measure seeks to provide a different view into the causes of uncertainty. We expect that each measure will have different utility depending on the current situation. Further, we seek to describe many of these measures as GVFs themselves. If we can do this it provides a single mechanism capable of capturing both our base sensorimotor signal predictions as well as the introspective measures we propose. This is appealing given that in a continual learning scenario we expect the GVF models to be incrementally constructed over time. If a single architecture is sufficient to represent both sensorimotor GVFs and their certainty measures then so too a single constructive algorithm may produce both. For the measures presented here we have been able to describe some as GVFs, but others remain unknown.

It is important to consider that TD methods, which we use in the GVF framework, specifically predict the *expected* return of a signal, not the return itself; GVFs form an expectation model. Thus, if we make some predictive statement such as "predicting joint angle" we are referring to the expectation of the return of the joint angle. In reality, the signal is a random variable that is supported over some distribution. With an expectation model, the predictions made may not reflect possible outcomes. However, the agent's predictive capabilities may be improved with additional state information. An alternative approach

would be to model the distribution over outcomes (Bellemare, Dabney, and Munos, 2017) and draw a prediction from that distribution. However, distributional models are not always practical or tractable.

For example, consider an agent that comes to a junction where it can choose between two doors, left or right. If it goes left its x position sensor reads -1 and if it goes left its x position sensor reads 1. On alternating days the agent chooses to go either right or left, thus, over all time there is an equal probability of choosing either door. A naive GVF predicting the position x at the junction would output a prediction of 0, which is not a valid outcome. However, if the agent's state includes which action was chosen on the previous day then it can predict exactly which door will be taken on each day.

How accurate have past predictions been?

Ultimately, the question we are asking is how certain we are that the given prediction will match the observed outcome. Thus, the most direct certainty measure is prediction accuracy or error. The other measures proposed in the following sections might be considered surrogate or supplemental.

Measures of prediction accuracy can be spread over the recency/state grid as shown in Figure 3.1. If considering a stateless representation then the question we are asking is "How accurate have the predictions been lately?" Such an approach was taken by White, Modayil, et al. (2014) wherein they use a trace on the TD-error to guide exploration towards surprising states, i.e., those states where prediction error was high. Their method, unexpected demon error (UDE) is given as:

$$U_t = \left| \frac{\bar{\delta}^{\beta}}{\sqrt{var[\delta] + \epsilon}} \right|. \tag{3.1}$$

In their equation $\bar{\delta}^{\beta}$ is an exponential moving average of the TD-error: $\bar{\delta}^{\beta}_{t+1} = (1-\beta)\bar{\delta}^{\beta}_{t} + \beta\delta_t$, where $\beta \in [0, 1]$. The term ϵ is a small constant used to prevent division by zero. For the denominator the variance of the TD-error is tracked from the beginning of time. In this way they can measure how much the current error compares to the usual variability. If the TD-error exceeds the usual variability there will be a spike in the UDE measure, which they take to indicate surprise. As the agent continues to improve its prediction the UDE measure will decrease. In using such a measure for certainty the idea is that if prediction accuracy has been good for recent predictions, then we should expect it to be good for the current

prediction as well, and vice-versa. This measure is studied in more detail in Chapter 4.

A state-based approach to prediction accuracy lets us ask the question, "When we were in this state before, how accurate were our predictions?" This would be useful for planning, and for anticipating state-regions of uncertainty. The state-based measure of error is only usable given that the agent has sufficient experience with a state-region of interest (the function approximation can be initialized to give high values indicating high uncertainty). A simple approach to this would use a GVF with the squared TD-error as the cumulant and $\gamma = 0$.

How much has the prediction output varied recently?

Consider the situation where an agent's predictions vary drastically in the recent past, say in the past few seconds. What might cause such behavior? One situation could be that the agent's internal representation is operating very close to a threshold. For example, this might happen in a function approximation setting where the inputs are binned into different ranges. Thus, this may indicate a need to improve the agent's representation.

Another cause could simply come from incomplete information. An agent in the real world is never able to observe the state of the universe directly. Agent's are limited to their locality and their sensory inputs. Thus, perception must be a process that unfolds over time. An agent gathers observations over time to construct its representation of relevant state information. This is illustrated by the duck-rabbit illusion shown in Figure 3.2. Most people are able to interpret this image as either a duck or a rabbit. Which is correct? With only this static image we cannot say. If instead we could move to observe the rest of the animal or see the animal itself move then we would be able to discern the truth. Consider the example of a cat on a lawn who perceives something that might be of danger. The cat might sink low and stare very hard at the potential threat, gathering more data until it decides if the threat exists or not. The cat is waiting for more information to validate the initial prediction. Other works have looked at using information gain as an exploration strategy for an RL agent (Oudeyer and Kaplan, 2009; Iwata et al., 2004; Storck et al., 1995).

How often has the current state been visited?

The agent learns from experience, thus to be able to make accurate predictions about a particular state it must have sufficient experience with that state. If visitation for a particular state is low then the agent might be hesitant about trusting predictions in that state. State

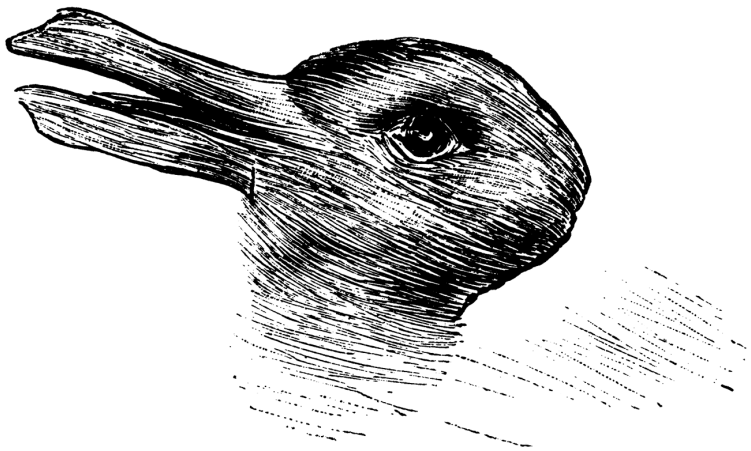


Figure 3.2: "Duck Rabbit Illusion" (1892)

visitation could also be useful for planning and decision making. In safe environments low state visitation might indicate an opportunity to explore and learn about the world (Machado, Bellemare, and Bowling, 2020). In dangerous environments the agent might prefer to stick with known states. Counting states may allow an agent to represent novelty and familiarity.

Counting state visitations in the tabular setting is straight forward, but with function approximation it is quite complicated. We have access to features, not states, so we might imagine simply counting features, but this is not the same. A particular configuration of features may not uniquely describe a state, or some features may be irrelevant to making accurate predictions. Further, we may want to share counting across related states. Being able to identify sets of features as relating to the same state is a complex task and beyond the scope of this short discussion. Tang et al. (2017) and Bellemare, Srinivasan, et al. (2016) discuss different approaches towards counting state-visitations.

Nonetheless, we present a naive approach to counting state visitations using a GVF. In this case we simply use a fixed cumulant of 1 and $\gamma = 0$. Then for a linear function approximator we would have a TD-error of

$$\delta_t = 1 - \mathbf{w}_t^\top \phi_t. \tag{3.2}$$

This predictor should be initialized to zero. In this setting a new state will have a count of 0 and a frequently visited state will be near 1. This approach would tend to over count

unchanging features. As well, it is best suited to binary features rather than real-valued ones. This measure is used in Chapter 4.

Thus far, we have only considered counting state visitations. The successor representation, which can be represented as a GVF, can be used to estimate counts along a trajectory of states (Machado, Bellemare, and Bowling, 2020).

In conclusion, counting state visitation can be phrased in many different ways and the correct way to do so is still an open question in the community.

How long has it been since this state was last visited?

This measure simply asks how long it has been since the agent last experienced a given state. This acts as a check on being overly confident if there has been considerable time since the agent last saw the state. If the environment is potentially changing then, despite good performance in the past, the agent may want to be wary of situations it has not seen in a long time. For a static environment this may not be an issue. However, with function approximation we know that if the agent does a lot of learning in one area of the state space, this can cause forgetting in other areas of the state space. In such a case the agent should not necessarily trust that its predictions are still good. It may need to relearn. This idea can also be extended to asking how long it has been since the agent followed a particular policy. Such measures might be implemented as a decaying trace over the features.

How predictable is the next state out of the current one?

If the state representation is sufficient then the next state should be predictable, within the bounds of the environment's stochasticity. Measuring such predictability could thus serve two purposes: 1) giving the agent an evaluation of its own representation, 2) giving the agent a certainty measure for decision making. If the predictability of the next state is low then the agent may wish to be hesitant about trusting its predictions and take action cautiously. The same information may already be captured in estimations of prediction error.

Have the predictions of the GVF converged?

Our learning algorithms typically take small steps towards their prediction target on each update. Even if we assume the simplest scenario where targets are unchanging, then a GVF will take several learning iterations to converge on its target. Thus, measuring the rate of

learning and convergence seems like a natural signal to consider. One method for measuring convergence is using RUPEE, which is tied to the GTD algorithm (White, 2015). GTD uses a second set of weights to capture part of the gradient of the mean squared projected Bellman error (MSPBE). This can be used to create an estimate of learning progress.

How up to date is our state information?

An agent operating in the real world has limited ability to observe the world and some of its state representation must be based on its memories of the past, e.g. "Where was the last place I saw my keys". If the observations those memories are based on are old then the agent might be hesitant to trust predictions based on that information. For example, consider a military general planning an offensive. If their information about enemy troop placement is a week old they will be less confident about planning an offensive than if the observation had just been made.

How far in the future is the prediction?

The prediction timescale might affect confidence. If we consider predictions in the more general sense (more general than GVFs and TD learning) then the timescale of a prediction can affect its certainty depending on what is being predicted. For example, as of the time of writing, I can predict with high confidence that tomorrow I will still be a graduate student and I can predict with high confidence that 2 years from now I will not be. However, I have a hard time making that same prediction for 6 months from now as it is uncertain exactly when I will complete the requirements of my degree.

On the other hand, when we consider the case of GVFs and TD learning there is not such a clear relationship between a prediction's timescale and its certainty. Samples of longer timescale returns typically have higher variability. However, GVFs predict the expectation of the target rather than the target itself. For static environments and policies the expectation is also static. Thus, the prediction timescale does not factor into the certainty of the expectation. Instead, it is the learning of the prediction that might be affected by the timescale. This effect should already be captured by other signals we have discussed, such as the measure of convergence.

How high is variance of the target signal?

The variance of the return will affect the predictor's ability to learn the expectation. As discussed, tracking convergence would help an agent to make decisions about trusting a given prediction, despite its variance. However, where variance could play a role is in choosing between different policies. In RL research, we often construct artificial domains and allow our algorithms to train until convergence, and what we care about is learning the best behavior in expectation. In risky environments this changes. Consider the case of autonomously driving a car, for which the agent receives more reward for arriving at its destination quickly and is penalized for accidents. Imagine an agent comparing two policies that both have the same expected return. The first policy drives sedately but consistently arrives safely at its destination. The second policy drives very quickly, but occasionally gets into accidents. We will presume that the two policies have the same expected return from their departure point, so from a standard RL perspective they are both equivalent. However, the second policy has much higher variability, and occasionally incurs damages. Thus, if an agent considered this variability in its decision making it may be able to discriminate between the two policies. This is discussed in more detail in Chapter 5 in which we demonstrate a method for estimating the variance of the return as a network of GVFs.

How reasonable is the predicted value?

The output of a predictor itself may be used to provide a measure of certainty. If a prediction falls outside the normal range of the target signal then the certainty should be lower depending on how far outside the range it falls. Such a measure can also be spread across the recency/state-based axes. The agent might consider the range of what it has seen over all-time, or it might consider the range of targets it's seen over the last hour. Additionally, the appropriate range might be dependent on the state, with different ranges being appropriate for different regions of state-space. Such a measure effectively acts as a sanity check.

3.4 Conclusion

This chapter connects GVFs with introspective measures and certainty measures. It serves as the motivation for the work presented in the next two chapters. While we have drawn attention to numerous introspective measures and consider the kinds of information they

may provide to an agent, in the limit we are not suggesting that such measures should be hand crafted. Instead, for our ideal continual learning agent, we imagine a mechanism that incrementally constructs its predictive GVF model over time. We would hope that such a mechanism would instead discover such signals on its own and suppose that some of these measures would capture the same information described in this chapter and the next. For a continual learning agent then, the key points being made are that the agent's knowledge construction framework should be able to use internally generated signals as prediction targets as well as capture statistical properties of its interactions and that the agent's state representation will be enhanced by including such measures. Ultimately, we believe this will allow the agent to make better decisions and learn general strategies for controlling its own learning process.

Chapter 4

Investigating Certainty Measures

Here we build off the ideas presented in Chapter 3. We provide several scenarios in which we demonstrate and evaluate the behavior of selected certainty measures.

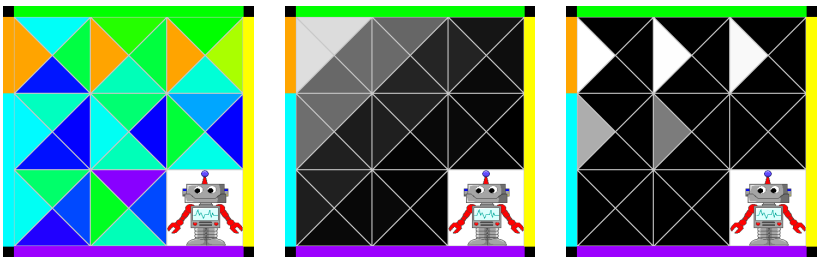
Section 4.1 first considers a simple robot in a grid world (Sherstan, Machado, White, et al., 2016). This demonstration shows that adding certainty measures provides additional information that might be useful for an agent. Further, it shows that multiple measures can be used together to provide richer information.

Section 4.2 demonstrates several certainty measures on a human controlled robot arm (Sherstan, Machado, Travnik, et al., 2017). The human moves the arm through a prespecified set of waypoints, but must choose whether to go left or right at a single decision point. This demonstration looks at how introspective measures might be used to detect states of uncertainty.

Finally, in Section 4.3 a robot learns a large collection of predictions about its observation space. Additionally, the robot produces a measure of surprise, UDE, and learns to predict UDE (Günther et al., 2018). Being able to predict such surprise might be key for avoiding damaging situations. Note that this work builds on preliminary work by Pilarski and Sherstan (2016).

4.1 Predicting Wall Hue in a Grid-World

In this section we provide a first illustrative example of how certainty measures might be used. The environment used for these experiments (Figure 4.1) is inspired by the "compass world", which was first used to demonstrate the representational power of using TD-nets with options (Sutton, Rafols, et al., 2006). In our experiments a simulated robot moves



(a) Prediction (b) Visitation (c) Error

Figure 4.1: A simulated robot in a grid world. See text for detailed explanation. In all cells the corresponding triangle opens in the direction of the prediction policy. For example, for **(a)** in cell (0,0) the orange triangle opening to the left indicates the hue the robot predicts it will see if it ran all the way to the left wall. **(a)** Prediction of wall hue (a continuous representation of color in [0,360)) in each of four directions. **(b)** State-action visitation. White is high visitation (high certainty). **(c)** Prediction of expected squared TD-error (related to variance). White indicates expectation of low error (high certainty).

through a 3x3 grid world and predicts wall hue (a continuous representation of color in [0,360)) in each of four directions (Figure 4.1a). The robot is only able to observe the hue of a wall when it "bumps" into that wall. For each direction the agent has one GVF that predicts the expected hue if it were to run all the way to the wall in that direction. The cumulant and timescale are both dependent on whether or not the agent observes the hue (takes action into the wall), *hue_present*, which is either 0 or 1. The timescale is defined as $\gamma_{t+1} = (1.0 - \mathit{hue\_present}_{t+1})$ and the cumulant is defined as $C_{t+1} = \mathit{hue\_present}_{t+1} * \mathit{hue}_{t+1}$. A tabular state representation is used.

At each timestep the hue of the walls may change. In this example, the hue of the upper wall has high variance while all other walls have low or no variance. Each cell of Figure 4.1a shows the predicted hue in each direction. The robot moves in a weighted random walk with a preference for moving up and left, as seen in the state-action visitation (Figure 4.1b). High certainty (white) corresponds to high visitation and low certainty (black) corresponds to low visitation (visitation is initialized to 0). In Figure 4.1c the robot makes predictions of the expected squared TD-error of the primary hue prediction. This prediction is made using another GVF that uses the squared TD-error of the hue GVF as its cumulant and $\gamma = 0$. High certainty (white) corresponds to low error and low certainty (black) corresponds to high error (error is initialized high).

Let us assume that the robot has some, as yet unspecified task, that depends on its ability to accurately predict hue. The robot can decide to only trust predictions in portions

of the world it has visited before, here the upper left. Further, when in the upper-left corner, the robot can see that despite high visitation it should not trust it's upward prediction as error (and variance) remain high. On the other hand, it can trust its leftward prediction as visitation is high and error is low. There are two main points here. The first is that adding these certainty measures to the state representation, something that is not typically done, provides additional information to the agent to improve its decision making abilities. Secondly, providing only a single measure was insufficient, visitation and error worked together to give a more detailed picture.

4.2 Decision Points on a Robot Arm

As previously stated, one of the motivations for my research has been to improve a prosthesis user's ability to perform everyday tasks with a powered prosthesis. In this section, we consider the case where a user performs a repetitive task with a robot arm. However, at one particular point of the task the agent can choose to move in either of two directions. We refer to this point as a *decision point*. We experimentally explore several introspective measures in this setting. These measures could be used to provide additional information to an agent which is not available in its sensorimotor stream. Further, these measures may allow an agent to discern decision points.

In this experiment, virtual waypoints are displayed on a screen (Figure 4.2b), which correspond to joint configurations of the shoulder rotation and elbow flexion joints of a robot arm (Figure 4.2a). A user guides the arm through these waypoints using a joystick (the colored trajectories are used as a legend for the figures that follow). Each trajectory starts at the bottom-middle waypoint (start) and proceeds upward. At the start of the experiment the user always goes left from the decision point (marked "split"). Around 4200 timesteps the user is permitted to choose to move the shoulder joint either left or right as they like.

As primary predictions, we use one GVF each to predict the joint positions of the shoulder and elbow joints. In the initial scenario we expect that certainty in these predictions should grow as the system learns to predict the positions of the joints. Once the user can choose between going left or right this split point becomes a point of uncertainty. No state information is provided that can tell the system which way the shoulder joint will go. Thus,

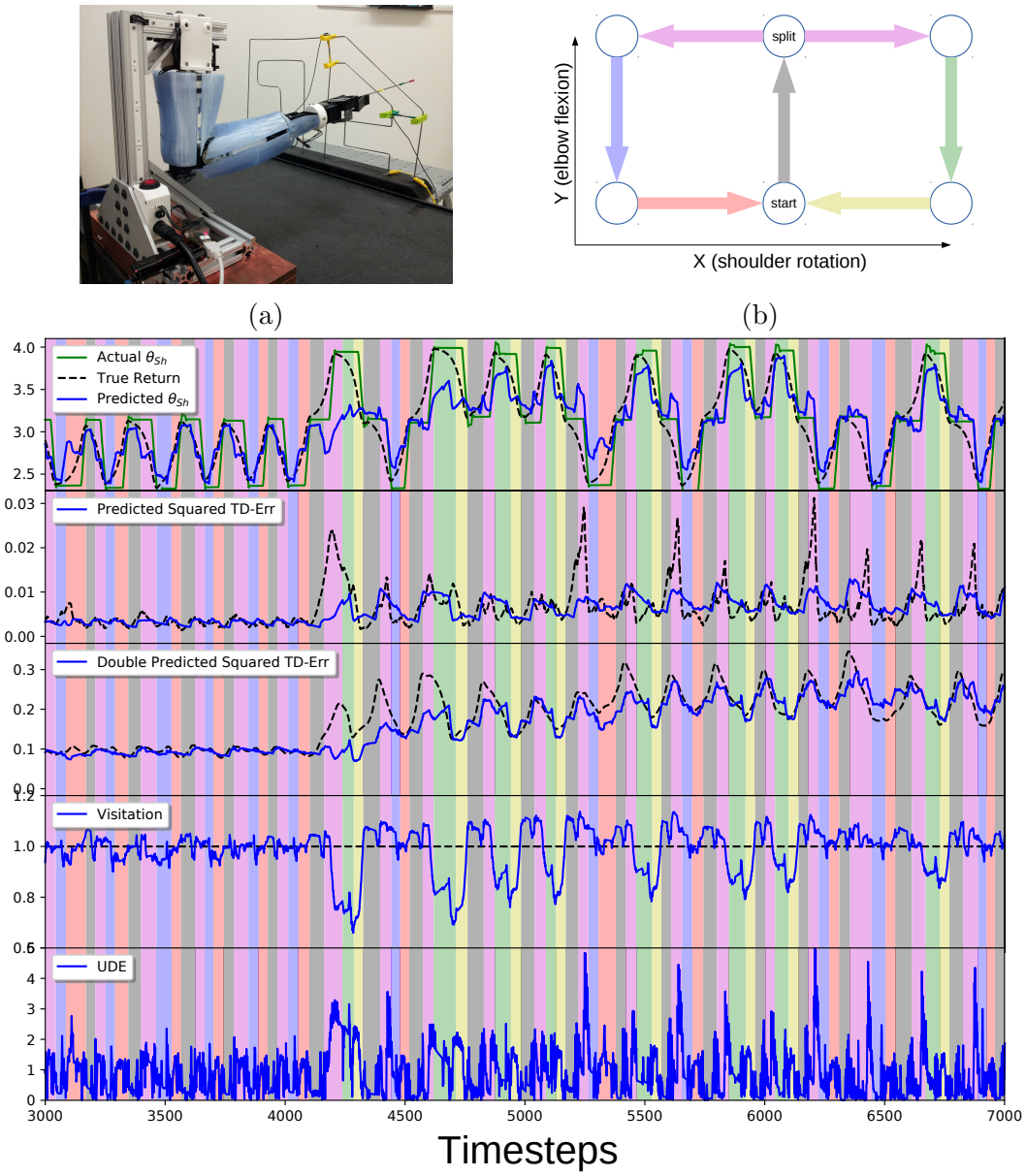


Figure 4.2: (**a**) The Bento Arm (**b**) A user controls the shoulder rotation (x) and elbow flexion (y) joints to move a pointer through a series of waypoints displayed onscreen. A circuit starts in the lower-middle waypoint and follows the arrows in the diagram. For the first portion of the experiment the user always goes the left, but after about 4000 timesteps they are free to choose to go either left or right. (**c**) Various measures relating to the shoulder rotation joint position. Shaded regions corresponds to the trajectories shown in Figure 4.2b. Note that the shading color changes once the corresponding waypoint is reached, movement will follow after. Predictions are shown in blue and the true return is shown in the dashed black. (**top**) Joint position. (**second**) Predicted prediction error. (**third**) Prediction of Predicted prediction error. (**fourth**) State visitation. (**bottom**) UDE, a measure of surprise.

we would expect that while certainty for the rest of the trajectory regions may grow, the region leading up to this split point will have low certainty and the predicted shoulder position will be more or less centered on the split waypoint until the user provides clarifying information by moving in one direction.

In this setting we make several GVF predictions, which are learned using GTD(λ) with $\lambda = 0.99$ (Equation (2.12)). The state representation is a fixed-length binary vector that is constructed by concatenating a bias unit and a 4-dimensional tile-coding of normalized shoulder angle ($\hat{\theta}_{Sh}$), elbow angle ($\hat{\theta}_{El}$), and decaying traces of the same (T_{Sh}, T_{El}), with a decay rate of 0.8 (Ex. $T_{Sh;t} = 0.8T_{Sh;t-1} + 0.2\hat{\theta}_{Sh;t}$). The angles were normalized over the effective joint range. These inputs were tile-coded (Sutton and Barto, 2018) with 100 coarse tilings of width 1.0, which were hashed to a size of 2048, giving a feature vector of length 2049 with 101 active features. Learning updates were made at a rate of 30 updates per second.

Additionally, we compute a simple method for counting state visitation, which is also represented as a GVF. For this measure the cumulant is always 1 and $\gamma = 0.0$ (Section 3.3.2). Finally, we compute UDE (Equation (3.1)) with $\beta = 0.6$.

We start by considering the prediction of the shoulder rotation, θ_{Sh}. For this GVF we use a cumulant of $(1 - \gamma)\theta_{Sh}$, which places the predicted return, and thus the prediction, in the same scale as θ_{Sh}. The prediction timescale is $\gamma = 0.967$ or approximately 1 s in the future.

Figure 4.2c (top) shows the position of the shoulder joint, the true return and the actual prediction made by the system. We see that prior to 4200 timesteps the trajectory was consistently to the left side and that the predictions match well against the ideal. After this point the predictions are initially off on the right side (green and yellow), but the system soon learns to predict these trajectories. The area of interest is the region approaching the split point (gray) and the region immediately following the split point. In the left only regime the GVF learns to make decent predictions in both regions, predicting movement to the left. Once the user begins choosing to go in either direction we see that in the gray region leading up to the split point the prediction tends to stay in the middle. Once the user switches joints we move into the pink region and here too the predictions tend towards the middle. It is only after the user begins moving the shoulder in one direction or the other that the predictions move to track the position of the shoulder joint. Also, in the gray and pink

regions, prior to movement, while generally predicting a central position there does appear to be a bias towards predicting in the direction that the user took the last time they were at the split point. This is to be expected from the tracking performed by online learning.

The remaining panels of Figure 4.2c show various certainty measures on this system. In the second panel the squared TD-error produced by the shoulder position estimator is used as the cumulant of a GVF. We see that the system has settled on relatively low error until the switch around 4200 timesteps at which point the system begins predicting high error for all states leading up to the split point (gray) and around the split point itself. Further, this prediction is itself used as a cumulant for another GVF in the third panel. Here we see that this predictor is able to anticipate high error much earlier. The bottom panel shows an estimate of visitation, which is also implemented as a GVF. Here we see that the system has relatively less experience with the right side of the task. As in Section 4.1, the predicted squared TD-error and the visitation could be taken together to explain whether or not a prediction should be trusted and may suggest why. Finally the UDE measure (White, 2015) shows that the agent gets spikes of surprise whenever the user starts moving left or right from the split point. These results suggest several measures that may indicate uncertainty in a GVF prediction. Using these approaches may highlight points where the trajectories diverge, such as they did here.

4.3 Predicting Surprise

Unexpected Demon Error (UDE), Equation (3.1), can be seen as a measure of surprise. It is computed by tracking the long-term and short-term statistics of TD-error and can thus be seen as an introspective measure. When UDE increases it indicates that the recent TD-errors are outside the standard deviation of the TD-errors over all time, thus such spikes are *surprising*. In this section we investigate UDE and predictions of UDE for a set of GVFs applied to the dataset of a robot arm.

For these experiments we use the Modular Prosthetic Limb (MPL v3) (Bridges et al., 2011), which is arguably one of the most advanced prosthetic arms to date. What sets the MPL apart as a prosthetic arm is its high degrees of freedom (17 active degrees) and its rich sensory stream. For each motor the robot provides sensory readings of load, position, temperature, and current. Additionally, each fingertip has a 3-axis accelerometer and 14-pad

Figure 4.3: The Modular Prosthetic Limb (MPL) used for the experiments. The arrows indicate the nature of the repeated disturbance imposed during the experiment. The green arrow indicates the direction of the provided perturbation, while the blue arrows indicate the resulting joint movement.

pressure sensor array to provide tactile sensation. Data packets are transmitted via UDP packets. For this experiment we only included sensory data from the motors.

This experiment consisted of the MPL being intermittently perturbed from a resting position. During the first two minutes of the experiment the arm remained motionless in its resting position. Note that this position is not passive, but is actively maintained by arm motors. This period allows the GVFs to learn predictions of the baseline behavior. Following this, the arm was perturbed by the experimenter at minute intervals as indicated in Figure 4.3. Following each perturbation the arm returned to its resting position. The total experimental data set spanned 5217 timesteps (approximately 20 minutes) with 21 perturbations. Outputs for position, velocity, load and temperatures are shown in Figure 4.4.

4.3.1 Predictive Architecture and Algorithmic Implementation

The experimental architecture, visualized in Figure 4.5, produces four sets of outputs: 1) bit level sensorimotor GVFs, 2) UDE for the sensorimotor GVFs, 3) predictions of UDE and 4) float level sensorimotor GVFs. As input to the first module, *GVF Module 1*, we use the raw bit representation of the data packet that is 3520 bits in length. *GVF Module 1* then makes a one-step prediction of each of these bits ($\gamma = 0$). For each GVF prediction the UDE is calculated with $\beta = 0.001$. Thus, *GVF Module 1* outputs 3520 predictions and 3520 UDE

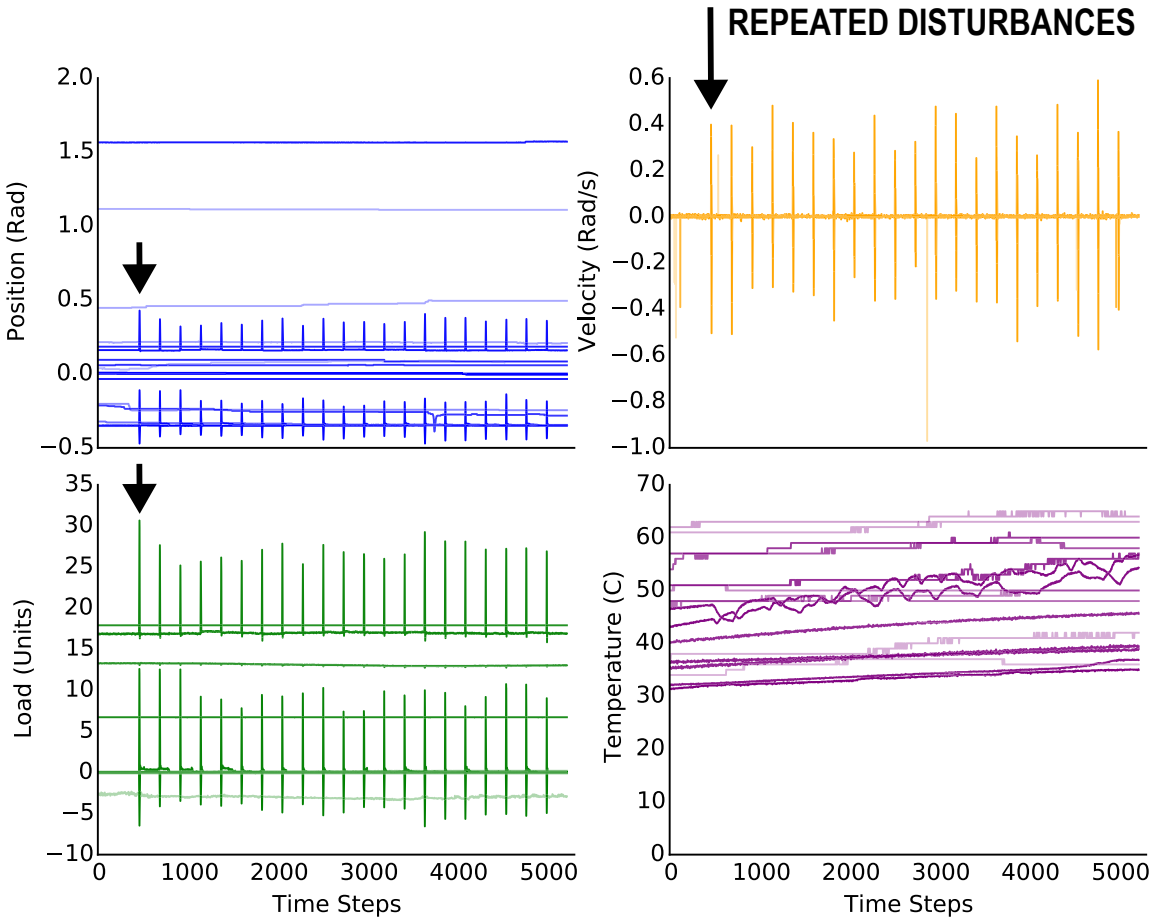


Figure 4.4: Decoded percept data from the robot over the 20 min duration of the experiment. The 21 disturbances are clearly identifiable in data from the position, velocity and the load sensors. The temperature sensors show an increasing temperature over the experiment, with additional increases for some sensors due to the perturbations.

values. *GVF Module 2* takes the first module's GVF predictions as its features and the UDE as its cumulant, thus predicting UDE. The discount rate for this module is $\gamma = 0.999$, which is very long. The final module, *GVF Module 3*, takes the first module's predictions as inputs and again predicts the UDP packet, but this time using the float representation that is 108 floats in length. A discount of $\gamma = 0.9$ is used. While this third module does not contribute to the narrative of this section it is included as an example of how predictions can be used hierarchically for increasingly complex predictive questions.

We chose to focus on predicting the sensorimotor stream as bits because in this form they provide examples of many different system dynamics. Some bits are completely stationary,

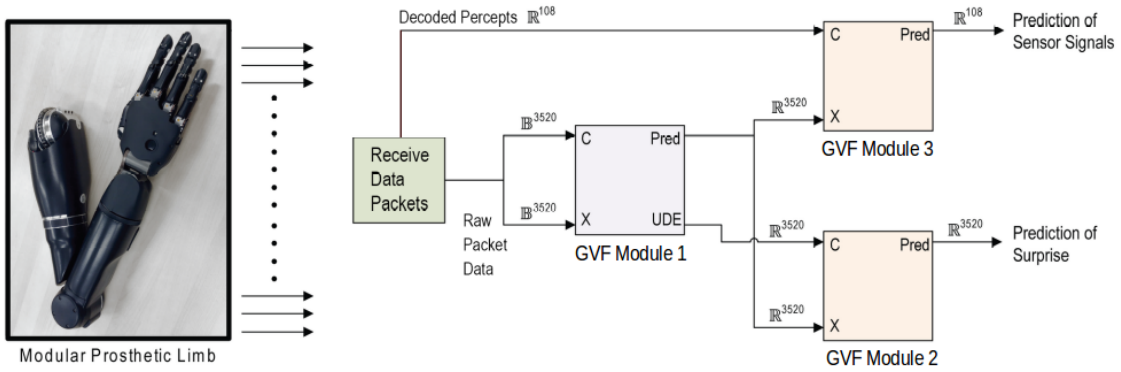


Figure 4.5: The prediction architecture used in the experiments. The sensor stream from the MPL on the left side is received over the network as a 3520 bit UDP packet, subsequently decoded into 108 floating point signals. These 3520 bits are delivered directly as both cumulant (C) and state (X) for Module 1. The output of the first Module is then fed into Module 2 and 3 as the state X used in predicting the 108 decoded sensor signals and also to make predictions about the UDE (surprise) of Module 1.

Bit/Line	Data / sensor readings
0-39	Header
40-903	Position
904-1767	Velocity
1768-2631	Load
2632-3495	Temperature
3496-3520	Footer & Checksum

Table 4.1: UDP packet structure: the position of data in the MPL binary percepts (each bit corresponding to the respective line in the figures that follow).

while others respond to the disturbance. The activity of some bits will change over time. Still other bits will change randomly. The categorization of the bit indexes is listed in Table 4.1.

All GVF predictions were represented using linear function approximation and learned using TD(λ) with accumulating traces and $\lambda = 0.99$ (see Equation (2.10)).

4.3.2 Experimental Results

Ideal Results

In order to provide intuition, we first created a simulated data set and computed the expected behaviors for each of the layers (Figure 4.6). Dark purple indicates highly active values while light blue indicates low activation. The simulated sensory bit stream is shown in subplot (a). It includes bits that do not change, bits that follow recurring patterns and others that respond

randomly. The first layer of GVF predictions (*GVF Module 1*) are shown in subplot (b). They predict the sensory bits at one step and therefore subplot (b) should match subplot (a) with a shift of one timestep. UDE is shown in subplot (c). UDE reacts differently to bits with different dynamics. After initial learning UDE should not react to random bits or bits that are constant. It should, however, react to bits that change in unexpected ways. This might happen if a bit suddenly turns on or if the frequency of activation exceeds what was previously observed. It will also occur for bits that turn on and off according to a recurring pattern, like those bits that respond to the disturbance, if the moving average of the UDE is short enough. Finally, the predictions of UDE are shown in subplot (d). Recall that these predictions use a very large discount that has a correspondingly long timescale. Thus, we expect that for bits with recurring patterns of UDE spiking our system will predict continually high UDE as this signal is repeatedly reinforced.

Bits and Bit Predictions

Figure 4.7 shows results for the real sensorimotor stream. Similarly to Figure 4.6, subplot (a) shows the raw bits of the UDP packets, (b) shows the one-step predictions from GVF Module 1, (c) shows UDE values and (d) shows predictions of UDE. The bits (along the y-axis) correspond to Table 4.1.

Some bits in the stream remain constant. This could be due to the bit representation of the float value or because the corresponding sensory data is itself unchanging. For example, readings from the hand should not change as it is not used in this experiment. Figure 4.8 provides a zoomed in look at Figure 4.7 for the position and velocity sensors (left) and the load sensors (right). We see that the position and velocity sensor's bits (Figure 4.8 (a, left)) are very constant with some fluctuations. However, around timestep 4090 we see a change in bit activation due to perturbation on the arm. Other bits, such as those for the load sensors (Figure 4.8 (a, right)), change continually. This may be the result of sensor noise or because some sensory reading is continually changing. For example, in order to maintain the resting position the arm must counter gravity, thus producing a continual, albeit fluctuating, load on the corresponding motors and resulting in slowly increasing temperature readings.

Predictions of constant signals are easy to learn and the one-step predictions (Figure 4.8 (b)) for these signals match the sensorimotor stream well. On the other hand, bits that change randomly or unpredictably (given the input features) are difficult or impossible to

predict and we instead expect the predictions to find the expectation over time, around 0.5. This is observed in Figure 4.8 (b, right) for some of the load bits between lines 2000 and 2050. For the position and velocity signals we see an increase in prediction magnitude around 4090, corresponding to the disturbance (Figure 4.8 (a, left)).

UDE and UDE Predictions

Our expectation is that UDE will show high activation at the beginning of learning until it has observed the baseline activity for the resting state. Afterwards UDE should go low and remain low until the TD-error increases due to the perturbations. This is clearly seen in Figure 4.7 (c). We see the repetitive pattern of UDE activation due to the perturbation. For example, we see this in lines 100, 950, 1800, which correspond to position, velocity and load respectively. Figure 4.8 (d, left) provides a closeup of one of these events. Of particular interest note that despite the regular noise of the velocity bits between lines 900 and 1000 in Figure 4.8 (a, left) the corresponding UDE signal only activates significantly during the disturbance around 4090. We can see this is due to the increased TD-errors in Figure 4.8 (c, left). Further, consider the highlighted UDE activation of the load sensors in Figure 4.8 (d, right). The corresponding TD-error alternates frequently (subplot (c)) following the perturbation at timestep 4310 and the UDE initially spikes and then tapers off its activation. It spikes again at 4460 in response to the negative TD-error.

Figure 4.7 (d), shows the predictions of UDE. These predictions do not significantly activate until after the first disturbance occurs. Following this, predictions for bits that are impacted by the disturbance remain high. The discount for these predictions is very high with $\gamma = 0.999$, which corresponds to roughly 1000 timesteps in expectation. Thus, these predictions are very long timescale. For bits that respond to each perturbation the prediction is reinforced and remains high. These can be interpreted as expectations that this bit will regularly be surprised. For other bits whose UDE spikes briefly the predictions of UDE have only a minimal, slow response, which might be treated as a filtering of UDE spikes. Further, for bits whose UDE is high (Figure 4.7 (c) around line 500 and 4090 timesteps) and then drops off the predicted UDE slowly decays.

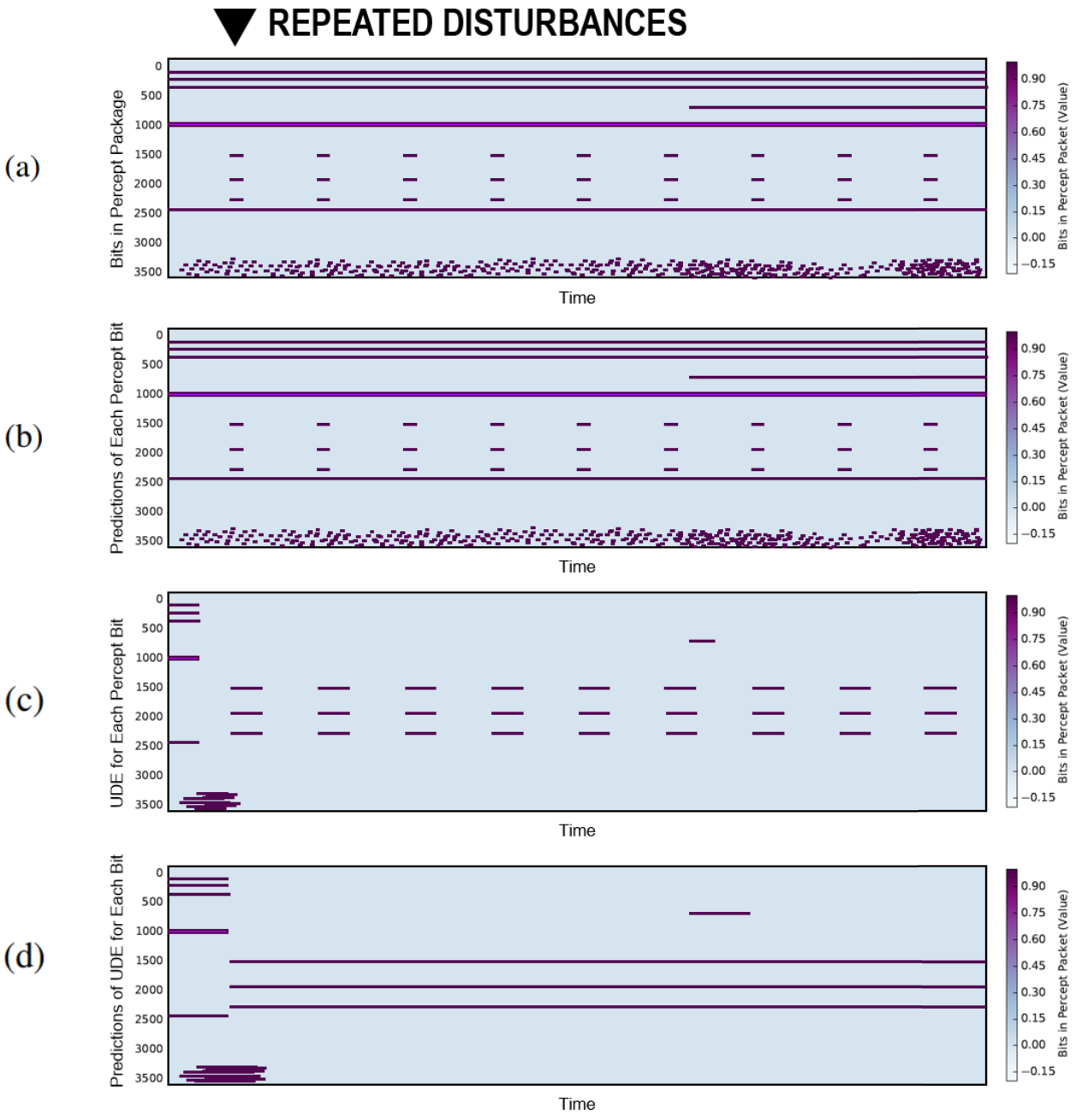


Figure 4.6: Simplified plots for the ideal relationship between (a) binary data, (b) predictions, (c) UDE and (d) predictions of UDE with $\gamma = 0.999$ for synthetic data.

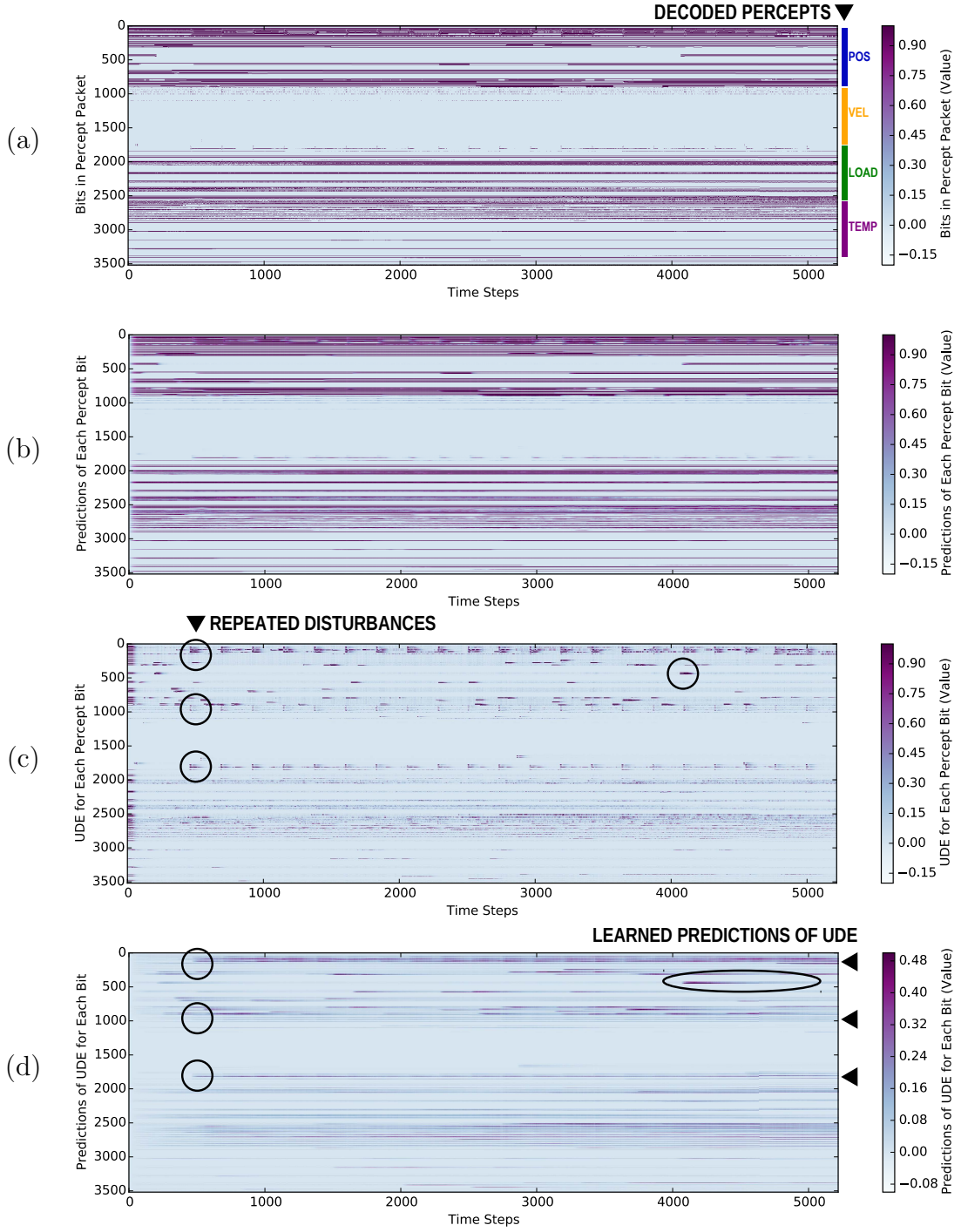


Figure 4.7: All recorded data for the experiment. The first subplot (a) shows the sensor stream from the MPL as decoded binaries. The second subplot (b) contains the myopic predictions for the binaries, provided by the first predictive layer. In the third subplot (c), the UDE is shown, followed by (d) the predictions about the UDE for a termination signal $\gamma = 0.999$.

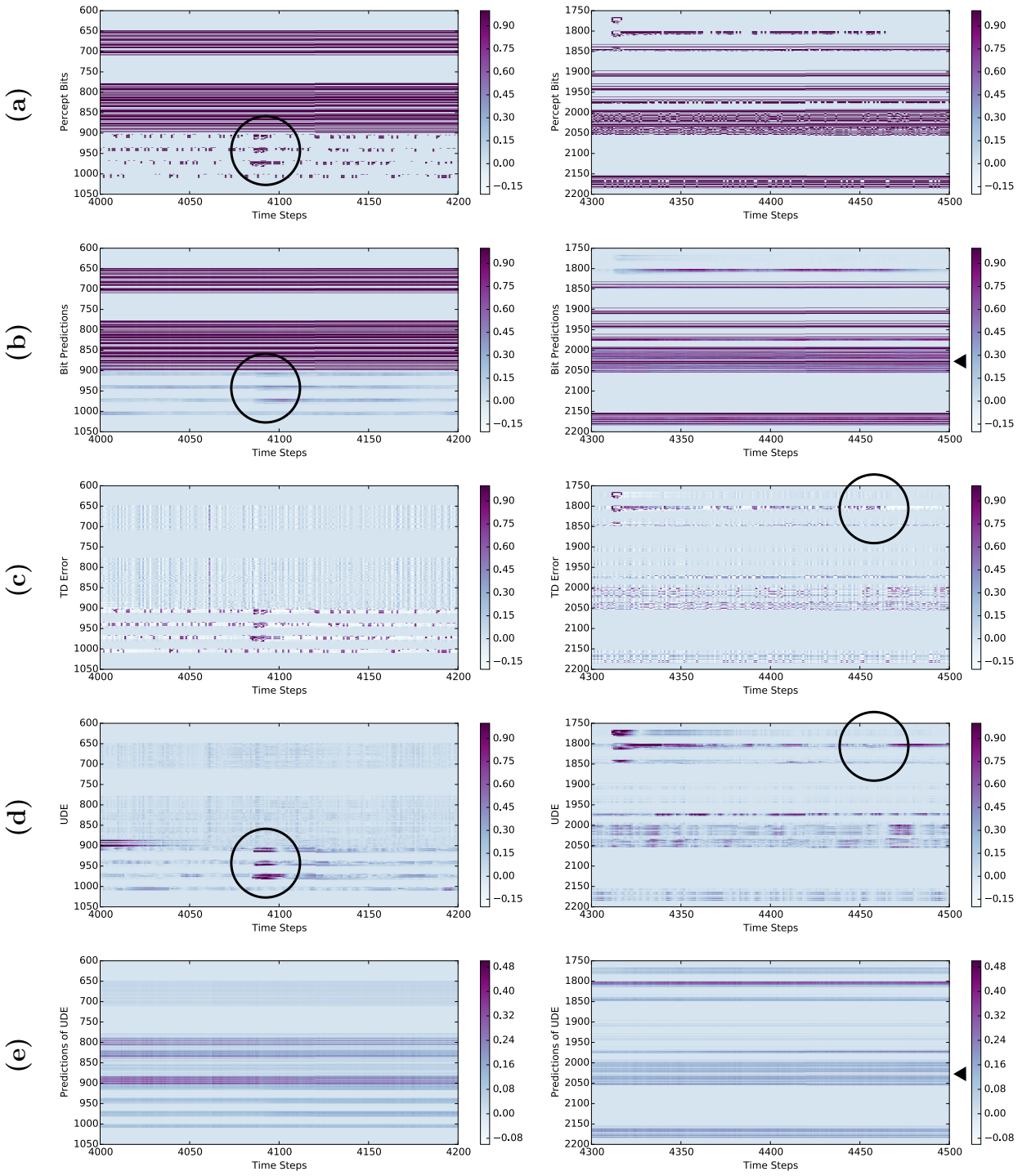


Figure 4.8: Zoomed in version of Figure 4.7. (a) Sensor data, (b) predictions ($\gamma = 0$), (c) prediction error, (d) UDE and (e) predictions of UDE ($\gamma = 0.999$) for (left) position and velocity sensors, (right) load sensors.

4.3.3 Discussion

This section explored the behavior of the UDE surprise measure and predictions of UDE in capturing information about the sensorimotor stream of a prosthetic arm. Here we focused on capturing information about the raw binary data of the MPL's data packets. The first layer in our architecture performed one-step predictions of the sensory bits. The primary role of these predictions was to produce TD-errors that were used for computing UDE. Additionally, these predictions were used as input features for predicting the UDE signals.

Using UDE allowed us to capture information about the temporal behavior of the bits. UDE essentially learns the baseline behavior of the signal and then spikes when something happens that is outside the norm. Thus, it was able to signify the effects of the perturbations on the position, velocity and load sensors. This provides insight into the sensory data that is not directly available in the sensory data itself. Predictions of UDE perform additional functions of filtering out temporary UDE spikes and identifying bits that regularly experience surprise. In this particular experimental setting the predictions of UDE are largely independent of state and instead capture whether or not a particular bit experiences frequent spikes in UDE. However, we would also expect to be able to make predictions of more state-dependent UDE, such as in Section 4.2.

UDE and predictions of UDE capture temporal information not directly available in the robot's data stream. Thus, providing such measures should enhance an agent's state representation, ultimately allowing it to learn to make better decisions. For example, spikes in UDE may occur when the robot is damaged. This piece of state information may allow the robot to adapt its behavior accordingly; if the robot learns to predict UDE, which occurs as the result of damage, it may be able to avoid such damage in the future. Or consider the setting of Section 4.2 in which there is one decision point in the trajectory that the user may take. A prediction of UDE may inform the robot arm that a decision point is approaching. This may enable the robot to take some sort of preemptive action depending on the task. For example, it may provide the user with relevant prompts.

4.4 Conclusion

This chapter built off of the ideas presented in Chapter 3. Specifically, we explored different introspective measures. We looked at how they could be used to capture information not

directly present in the observation stream. These measures capture temporal information over the agent's past experience. Such information, we speculate, will enhance an agent's representation of state and ultimately enable it to make better decisions.

In Section 4.1 we visualized visitation counting and prediction error and suggested that these could enhance an agent's representation. These suggest that such introspective measures add information that was not already present in the observation stream and further showed how multiple measures could work together. Section 4.2 presented a human-robot interaction scenario in which the human was faced with a single, state-dependent, point of decision as it controlled the robot arm. We considered predictions of error, visitation counts and surprise and suggested that these might be useful for identifying and predicting decision points—points of uncertainty. Finally, in Section 4.3 we conducted another demonstration on a robot arm. In this setting the agent learned to make predictions of its sensorimotor stream under normal behavior. The arm was then regularly perturbed and the agent used error-based surprise (UDE, Equation (3.1)), and predictions of surprise to capture information about these perturbations. This work suggested how predictions of surprise might be used for predicting deviations from normal behavior (e.g., predicting damage).

Taken as a whole, these three demonstrations further our conjecture that introspective measures can be a source of useful information for agents that continually, and autonomously, learn and adapt in real-world environments. In the next chapter we present a new method and detailed experimental analysis for estimating one specific introspective measure, the variance of the return.

Chapter 5

Using GVFs to Learn the Variance of the Return

In Chapter 3, we conjectured that introspective measures would be useful as part of an agent's state representation, and, further, that some such measures could be captured as GVFs. In this chapter we show that a sequential pair of GVFs can be used to directly estimate the variance of the return—an inherently introspective measure. The key to our approach is to use the squared TD-error of the first TD estimator as the cumulant of the second. Additionally, the second GVF uses the square of the first GVF's discount as its own discount.

In this chapter, we describe our algorithm and present theoretical results in the tabular-state setting. We experimentally evaluate our algorithm alongside an existing, but indirect method for estimating the variance of the return. Most of the experiments are performed in the tabular setting, but we also demonstrate our algorithm with linear function approximation. Our method is simpler than the indirect approach, and our experimental results show that it performs just as well as the indirect method, but is generally more robust.

The body of this chapter, and the original paper it is based on, motivates this research towards more traditional reinforcement learning issues, such as risk-sensitive RL, exploration, and parameter adaptation. However, our own motivation for pursuing this line of research was based on the premises posed in Chapter 3. The variance of the return may be useful as a certainty measure, providing an agent with additional information about its environment and learning process and allowing the agent to choose between different policies.

As a note to the reader, in keeping with the notation used by other works in this area, throughout this chapter we use J to indicate the value estimate and V to indicate variance.

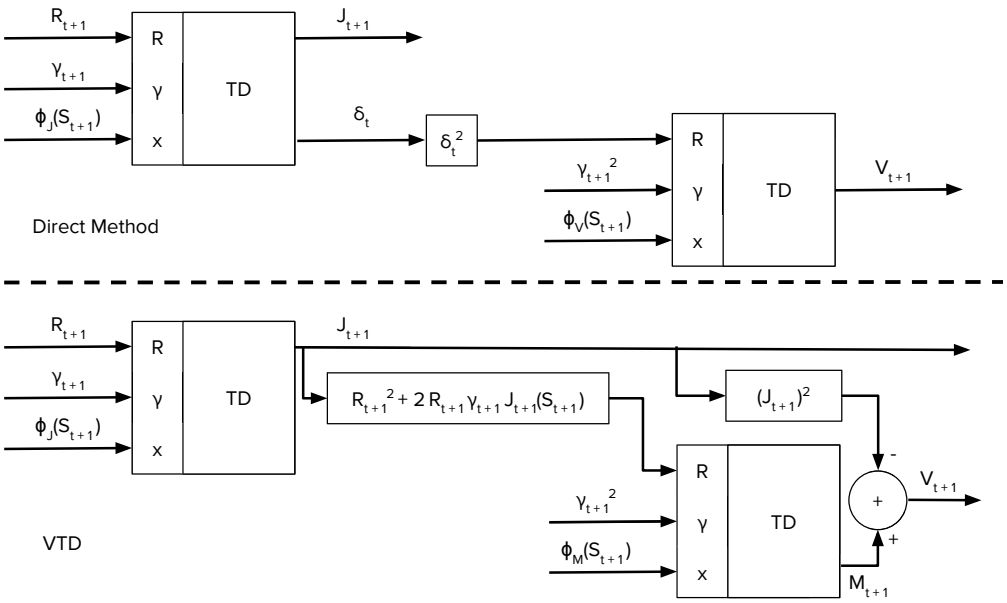


Figure 5.1: Each TD node takes as input a cumulant C, a discounting function γ, and features ϕ. For the direct method (**top**) the squared TD-error of the first-stage value estimator is used as the meta-reward for the second-stage V estimator. For VTD (**bottom**), a more complex computation is used for the meta-reward and an extra stage of computation is required.

5.1 Introduction

Conventionally, in reinforcement learning (RL) the agent estimates the expected value of the return—the discounted sum of future rewards—as an intermediate step to finding an optimal policy. The agent estimates the value function by averaging the returns observed from each state in a trajectory of experiences. To estimate this value function online—while the trajectory is still unfolding—we update the agent's value estimates towards the expected return. Algorithms that estimate the expected value of the return in this way are called temporal-difference (TD) learning methods. However, it is reasonable to consider estimating other functions of the return beyond the first moment. For example, Bellemare, Dabney, and Munos (2017) estimated the distribution of returns explicitly. In this chapter, we focus on estimating the variance of the return using TD methods.

The variance of the return can be used to design algorithms that account for risk in decision making. The main approach is to formulate the agent's objective as maximizing reward, while minimizing the variance of the return (Sato et al., 2001; Prashanth and Ghavamzadeh, 2013; Tamar, Di Castro, et al., 2012).

An estimate of the variance of the return can also be useful for adapting the parameters of a learning system automatically, thus avoiding time-consuming, human-driven meta parameter tuning. Sakaguchi and Takano (2004) used the variance estimate explicitly in the decision making policy to set the temperature variable in the Boltzmann action selection rule. Using variance in this way can automatically adjust the amount of exploration, allowing the learning system to adapt to new circumstances online. Conventionally, this temperature would either be set to a constant or decayed according to a fixed schedule. In either circumstance, the performance can be quite poor in non-stationary domains, and a human expert is required to select the constant value or fixed schedule. Similarly, White and White (2016) estimated the variance of the return to automatically adapt the trace-decay parameter, λ, used in learning updates of TD algorithms (see Section 5.2 for an explanation of the role of λ). Not only does this approach avoid the need to tune λ by hand, but it can result in faster learning.

The variance V of the return can be estimated either directly or indirectly. Indirect estimation involves computing an estimate of variance from estimates of the first and second moments. Sobel (1982) was the first to formulate Bellman operators for the second moment and showed how this could be used to compute variance indirectly. This is the approach used by Tamar, Castro, et al. (2016), Tamar and Mannor (2013), and Prashanth and Ghavamzadeh (2013). White and White (2016) introduced several extensions to this indirect method including estimation of the λ-return (Sutton and Barto, 2018), support for off-policy learning (Sutton, Maei, et al., 2009; Maei, 2011), and state-dependent discounting (Sutton, Modayil, et al., 2011; White, 2017). Their method, which they refer to as VTD, serves as the indirect estimation algorithm used in this chapter. We note that an alternative method, which we do not investigate here, could be to estimate the distribution of returns as done by Bellemare, Dabney, and Munos (2017) and compute the variance from this estimated distribution.

Variance may also be estimated directly. Tamar, Di Castro, et al. (2012) gave a direct algorithm but restricted it to estimating cost-to-go returns in a strictly episodic manner, i.e., estimates are only updated after an entire trajectory has been captured. We introduce a new algorithm for directly estimating the variance of the return incrementally using TD methods. Our algorithm uses two TD learners, one for estimating value and the other for estimating the variance of the return. These estimators operate in series with the squared

TD-error of the value learner serving as the reward of the variance learner and the squared discount rate of the value learner serving as the discount rate of the variance learner. Like VTD (White and White, 2016), our algorithm supports estimating the variance of the λ-return, state-dependent discounting, estimating the variance of the on-policy return from off-policy samples, and estimating the variance of the off-policy return from on-policy samples (Section 5.3.2 motivates these extensions). We call our new algorithm Direct Variance TD (DVTD). We recognize that the algorithm of Sato et al. (2001) can be seen as the simplest instance of our algorithm, using the on-policy setting with fixed discounting and no traces[1]. Sakaguchi and Takano (2004) also used this simplified algorithm, but treated the discount of the variance estimator as a free parameter.

We introduce a Bellman operator for the variance of the return, and further prove that, even for a value function that does not satisfy the Bellman operator for the expected return, the error in this recursive formulation is proportional to the error in the value function estimate. Interestingly, the Bellman operator for the second moment requires an unbiased estimate of the return (White and White, 2016). Since our Bellman operator for the variance avoids this term, it has a simpler update. As shown in Figure 5.1, Both DVTD and VTD can be seen as a network of two TD estimators running sequentially. Note that we restrict our formal derivations and subsequent analysis to the table lookup setting.

Our primary goal is to understand the empirical properties of the direct and indirect approaches. In general, we found that DVTD is just as good as VTD and in many cases better. We observe that DVTD behaves better in the early stages of learning before the value function has converged. Furthermore, we observe that the variance of the estimates of V can be higher for VTD under several circumstances: (1) when there is a mismatch in step-sizes between the value estimator and the V estimator, (2) when traces are used with the value estimator, (3) when estimating V of the off-policy return, and (4) when there is error in the value estimate. Finally, we observe significantly better performance of DVTD in a linear function approximation setting. Overall, we conclude that the direct approach to estimating V, DVTD, is both simpler and better behaved than VTD.

[1]Dimitrakakis (2006) used a related TD method, which estimates the squared TD-error

Table 5.1: Algorithm Notation

J	estimated value function of the target policy π.
M	estimate of the second moment.
V	estimate of the variance.
C, $\bar{C}$	meta-reward used by the J and (M, V) estimators.
λ	bias-variance parameter of the target λ-return.
κ, $\bar{\kappa}$	trace-decay parameter of the J and (M, V) estimators.
γ, $\bar{\gamma}$	discounting function used by J and (M, V) estimators.
δ_t, $\bar{\delta}_t$	TD-error of the J and (M, V) estimators at time t.
$\bar{\rho}$	importance sampling ratio for estimating the variance of the target return from off-policy samples.
η	importance sampling ratio used to estimate the variance of the off-policy return.

5.2 The MDP Setting

We model the agent's interaction with the environment as a finite Markov decision process (MDP) consisting of a finite set of states $\mathcal{S}$, a finite set of actions, $\mathcal{A}$, and a transition model $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ defining the probability $p(s'|s, a)$ of transitioning from state s to s' when taking action a. In the policy evaluation setting considered in this chapter, the agent follows a fixed policy $\pi(a|s) \in [0, 1]$ that provides the probability of taking action a in state s. At each timestep the agent receives a random cumulant C_{t+1} (typically reward R), dependent only on S_t, A_t, S_{t+1}. The return is the discounted sum of future rewards

$$\begin{aligned} G_t &= C_{t+1} + \gamma_{t+1} C_{t+2} + \gamma_{t+1}\gamma_{t+2} C_{t+3} + \dots \\ &= C_{t+1} + \gamma_{t+1} G_{t+1}. \end{aligned} \tag{5.1}$$

where $\gamma \in [0, 1]$ specifies the degree to which future rewards are discounted. Note that we define discounting as state-dependent such that $\gamma_{t+1} \equiv \gamma(S_{t+1})$. This allows us to combine the specification of continuing and episodic tasks. Further implications of this are discussed in Section 5.3.2.

The value of a state, $j(s)$, is defined as the expected return from state s under a particular policy π:

$$j(s) = \mathbb{E}_\pi[G_t | S_t = s]. \tag{5.2}$$

We use j to indicate the true value function and J the estimate. The TD-error is the

difference between the one-step approximation and the current estimate:

$$\delta_t = C_{t+1} + \gamma_{t+1} J_t(S_{t+1}) - J_t(S_t). \tag{5.3}$$

This can then be used to update the value estimator using a TD method, such as TD(0) as follows:

$$J(s)_{t+1} = J(s)_t + \alpha \delta_t \tag{5.4}$$

5.3 Estimating the Variance of the Return

For clarity of presentation, we first discuss the simplest version of both the direct and indirect methods and present the full algorithms in Section 5.3.2.

The direct TD method uses both a value estimator and a variance estimator. The *value estimator* provides an estimate of the expected return. The *variance estimator*, on the other hand, uses the value estimator to provide an estimate of the variance of the return. Since we use TD methods for both the value and variance estimators we need to adopt additional notation; variables with a bar are used by either the second moment or variance estimator. Otherwise, they are used by the value estimator. The key to using both the indirect and direct methods as TD methods is to provide a discounting function, $\bar{\gamma}$, and a meta-reward, $\bar{C}$. In the following, we present a simplified TD(0) version of both algorithms.

Simplified Direct Variance Algorithm

$$\begin{aligned} \bar{\gamma}_{t+1} &\leftarrow \gamma_{t+1}^2 \\ \bar{C}_{t+1} &\leftarrow \delta_t^2 \\ \bar{\delta}_t &\leftarrow \bar{C}_{t+1} + \bar{\gamma}_{t+1} V_t(s') - V_t(s) \\ V_{t+1}(s) &\leftarrow V_t(s) + \bar{\alpha} \bar{\delta}_t \end{aligned} \tag{5.5}$$

Simplified Second Moment Algorithm

$$\begin{aligned} \bar{\gamma}_{t+1} &\leftarrow \gamma_{t+1}^2 \\ \bar{C}_{t+1} &\leftarrow C_{t+1}^2 + 2\gamma_{t+1} C_{t+1} J_{t+1}(s') \\ \bar{\delta}_t &\leftarrow \bar{C}_{t+1} + \bar{\gamma}_{t+1} M_t(s') - M_t(s) \\ M_{t+1}(s) &\leftarrow M_t(s) + \bar{\alpha} \bar{\delta}_t \\ V_{t+1}(s) &\leftarrow M_{t+1}(s) - J_{t+1}(s)^2 \end{aligned} \tag{5.6}$$

5.3.1 Derivation of the Direct Method

We now derive the direct method for estimating the variance of the return. For clarity, we start with the simple case described in Section 5.3. Section 5.3.2 presents the more general extended algorithm.

The derivation of the direct method follows from characterizing the Bellman operator for the variance of the return: Theorem 1 gives a Bellman equation for the variance v. It has the form of a TD target with meta-reward $\bar{C}_t = \delta_t^2$ and discounting function $\bar{\gamma}_{t+1} = \gamma_{t+1}^2$. Therefore, we can estimate V using TD methods.

The Bellman operators for the variance are general, in that they allow for either the episodic or continuing setting, by using variable γ. By directly estimating variance, we avoid a second term in the cumulant that is present in approaches that estimate the second moment (Tamar and Mannor, 2013; Tamar, Castro, et al., 2016; White and White, 2016).

To have a well-defined solution to the fixed point, we need the discount to be less than one for some transition (White, 2017; Yu, 2015). This corresponds to assuming that the policy is proper, for the cost-to-go setting (Tamar, Castro, et al., 2016).

Assumption 1. *The policy reaches a state s where $\gamma(s) < 1$ in a finite number of steps.*

Lemma 1. *For $j(s) = \mathbb{E}\left[G_{t+1}|S_t = s\right]$, i.e., satisfying the Bellman function $b : \mathcal{S} \times \mathcal{A} \times \mathbb{R} \times \mathcal{S} \to \mathbb{R}$,*

$$\mathbb{E}[b(S_t, A_t, R_{t+1}, S_{t+1})(G_{t+1} - j(S_{t+1}))|S_t = s] = 0$$

Proof. Let $b_t = b(S_t, A_t, R_{t+1}, S_{t+1})$. By the law of total expectation:

$$\mathbb{E}\left[b_t(G_{t+1} - j(S_{t+1}))|S_t = s\right] = \mathbb{E}\left[\mathbb{E}[b_t(G_{t+1} - j(S_{t+1}))|S_t, A_t, R_{t+1}, S_{t+1}]|S_t = s\right]$$

Given S_t, A_t, R_{t+1} and S_{t+1}, b_t is constant and can be moved outside of the expectation. Therefore,

$$\begin{aligned}
&\mathbb{E}[b_t(G_{t+1} - j(S_{t+1}))\Big|S_t, A_t, R_{t+1}, S_{t+1}] \\
&\quad = \mathbb{E}\left[b_t\Big|S_t, A_t, R_{t+1}, S_{t+1}\right] \times \mathbb{E}\left[G_{t+1} - j(S_{t+1})\Big|S_t, A_t, R_{t+1}, S_{t+1}\right]
\end{aligned}$$

Because

$$\mathbb{E}\left[G_{t+1} - j(S_{t+1})\Big|S_t, A_t, R_{t+1}, S_{t+1}\right] = 0$$

the result follows. □

Theorem 1. *For any $s \in \mathcal{S}$,*

$$\begin{aligned} j(s) &= \mathbb{E}\left[R_{t+1} + \gamma_{t+1} j(S_{t+1}) \mid S_t = s\right] \\ v(s) &= \mathbb{E}\left[\delta_t^2 + \gamma_{t+1}^2 v(S_{t+1}) \mid S_t = s\right] \end{aligned} \tag{5.7}$$

Proof. First we expand $G_t - j(S_t)$, from which we recover a series with the form of a return.

$$\begin{aligned} G_t - j(S_t) &= R_{t+1} + \gamma_{t+1} G_{t+1} - j(S_t) \\ &= R_{t+1} + \gamma_{t+1} j(S_{t+1}) - j(S_t) + \gamma_{t+1}(G_{t+1} - j(S_{t+1})) \\ &= \delta_t + \gamma_{t+1}(G_{t+1} - j(S_{t+1})) \end{aligned} \tag{5.8}$$

The variance of G_t is therefore

$$\begin{aligned} v(s) &= \mathbb{E}\left[\left(G_t - \mathbb{E}\left[G_t \mid S_t = s\right]\right)^2 | S_t = s\right] \\ &= \mathbb{E}\left[\left(G_t - j(s)\right)^2 \mid S_t = s\right] \\ &= \mathbb{E}\left[\left(\delta_t + \gamma_{t+1}(G_{t+1} - j(S_{t+1}))\right)^2 \mid S_t = s\right] \\ &= \mathbb{E}\left[\delta_t^2 \mid S_t = s\right] + \mathbb{E}\left[\gamma_{t+1}^2 (G_{t+1} - j(S_{t+1}))^2 \mid S_t = s\right] \\ &\quad + 2\mathbb{E}\left[\gamma_{t+1}\delta_t (G_{t+1} - j(S_{t+1})) \mid S_t = s\right] \end{aligned} \tag{5.9}$$

Equation (5.7) follows from Lemma 1 which shows
$\mathbb{E}\left[\gamma_{t+1}\delta_t(G_{t+1} - j(S_{t+1})) \mid S_t = s\right] = 0$. Similar to Lemma 1, using the law of total expectation, $\mathbb{E}\left[\gamma_{t+1}^2 (G_{t+1} - j(S_{t+1}))^2 \mid S_t = s\right] = \mathbb{E}\left[\gamma_{t+1}^2 v(S_{t+1}) \mid S_t = s\right]$.

□

We provide an initial characterization of error in the variance estimate obtained under this recursion, when an approximate value function rather than the true value function is used. As we show in Theorem 2, the resulting error in the variance estimator is proportional to the squared error in the value estimate, and discounted accumulated errors into the future. If the approximation error is small, we expect this accumulated error to be small, particularly as the accumulation errors are signed and so can cancel, and because they are discounted. However, more needs to be done to understand the impact of this accumulated error.

Theorem 2. *For approximate value function J with variance estimate*
$V(s) = \mathbb{E}\left[\delta_t^2 + \gamma_{t+1}^2 V(S_{t+1}) \,|S_t = s\right]$*, if there exists $\epsilon : \mathcal{S} \to [0, \infty)$ bounding squared value*

estimation error $(J(s)-j(s))^2 \leq \epsilon(s)$ *and accumulation error* $|\mathbb{E}[\gamma_{t+1}\delta_t(j(S_{t+1})-J(S_{t+1}))+\gamma_{t+1}^2\gamma_{t+2}\delta_{t+1}(j(S_{t+2})-J(S_{t+2}))+\ldots\,|S_t=s]| \leq \epsilon(s)$*, then*

$$\left|v(s)-\mathbb{E}\left[\delta_t^2+\gamma_{t+1}^2V(S_{t+1})\mid S_t=s\right]\right|\leq 3\epsilon(s)$$

Proof. We can re-express the true variance in terms of the approximation J, as

$$\begin{aligned}v(s)&=\mathbb{E}\left[(G_t-j(s)+J(s)-J(s))^2\mid S_t=s\right]\\&=\mathbb{E}\left[(G_t-J(s))^2\mid S_t=s\right]+(J(s)-j(s))^2\\&\quad+2\mathbb{E}\left[G_t-J(s)\mid S_t=s\right](J(s)-j(s))\end{aligned}\tag{5.10}$$

This last term simplifies to

$$\begin{aligned}\mathbb{E}\left[G_t-J(s)\mid S_t=s\right]&=\mathbb{E}\left[G_t-j(s)\mid S_t=s\right]+j(s)-J(s)\\&=j(s)-J(s)\end{aligned}\tag{5.11}$$

giving $(J(s)-j(s))^2+2(j(s)-J(s))(J(s)-j(s))=-(J(s)-j(s))^2$. We can use the same recursive form as (5.9), but with J, giving

$$\begin{aligned}\mathbb{E}\left[(G_t-J(s))^2\mid S_t=s\right]=&\mathbb{E}\left[\delta_t^2+\gamma_{t+1}^2V(S_{t+1})\mid S_t=s\right]\\&+2\mathbb{E}\left[\gamma_{t+1}\delta_t(G_{t+1}-J(S_{t+1}))\mid S_t=s\right]\\&+2\mathbb{E}\left[\gamma_{t+1}^2\gamma_{t+2}\delta_{t+1}(G_{t+2}-J(S_{t+2}))|S_t=s\right]+\ldots\end{aligned}\tag{5.12}$$

where the terms involving $\delta_t(G_{t+1}-J(S_{t+1}))$ accumulate. Notice that

$$\begin{aligned}\left|\mathbb{E}\left[\gamma_{t+1}\delta_t(G_{t+1}-J(S_{t+1}))\mid S_t=s\right]\right|=&\left|\mathbb{E}\left[\gamma_{t+1}\delta_t(G_{t+1}-j(S_{t+1}))\mid S_t=s\right]\right.\\&\left.+\,\mathbb{E}\left[\gamma_{t+1}\delta_t(j(S_{t+1})-J(S_{t+1}))\mid S_t=s\right]\right|\\=&\left|\mathbb{E}\left[\gamma_{t+1}\delta_t(j(S_{t+1})-J(S_{t+1}))\mid S_t=s\right]\right|\end{aligned}$$

where the second equality follows from Lemma 1. By the same argument as in Lemma 1, this will also hold true for all the other terms in (5.12). By assumption, the sum of all these covariance terms between j and J are bounded by $\epsilon(s)$. Putting this together, we get

$$\begin{aligned}|v(s)-V(s)|&=\left|v(s)-\mathbb{E}\left[\delta_t^2+\gamma_{t+1}^2V(S_{t+1})\mid S_t=s\right]\right|\\&\leq 2\epsilon(s)+(J(s)-j(s))^2\leq 3\epsilon(s)\end{aligned}$$

□

5.3.2 The Extended Direct Method

Here, we extend the direct method to support estimating the λ-return, state-dependent γ, eligibility traces and off-policy estimation, just as White and White (2016) did with VTD. We first explain each of these extensions before providing our full direct algorithm and VTD.

The λ-return is defined as

$$G_t^\lambda = C_{t+1} + \gamma_{t+1}(1-\lambda_{t+1})J_t(S_{t+1}) + \gamma_{t+1}\lambda_{t+1}G_{t+1}^\lambda$$

and provides a bias-variance trade-off by incorporating J, which is a potentially lower-variance but biased estimate of the return. This trade-off is determined by a state-dependent trace-decay parameter, $\lambda_t \equiv \lambda(S_t) \in [0,1]$. When $J_t(S_{t+1})$ is equal to the expected return from $S_{t+1} = s$, then $\mathbb{E}_\pi[(1-\lambda_{t+1})J_t(S_{t+1}) + \gamma_{t+1}\lambda_{t+1}G_{t+1}^\lambda | S_{t+1} = s] = \mathbb{E}_\pi[G_{t+1}^\lambda | S_{t+1} = s]$, and so the λ-return is unbiased. Beneficially the expected value $J_t(S_{t+1})$ is lower-variance than the sample G_{t+1}^λ. If J_t is inaccurate, however, some bias is introduced. Therefore, when $\lambda = 0$, the λ-return is lower-variance but can be biased. When $\lambda = 1$, the λ-return equals the Monte Carlo return (Equation (5.1)); in this case, the update target exhibits more variance, but no bias. In the tabular setting evaluated in this chapter, λ does not affect the fixed point solution of the value estimate, only the rate at which learning occurs. It does, however, affect the observed variance of the return, which we estimate. The λ-return is implemented using traces as in the following TD(λ) algorithm, shown with accumulating traces:

$$\begin{aligned} \mathbf{z}_t(s) &\leftarrow \begin{cases} \gamma_t\lambda_t\mathbf{z}_{t-1}(s) + 1 & s = S_t \\ \gamma_t\lambda_t\mathbf{z}_{t-1}(s) & \forall s \in \mathcal{S}, s \neq S_t \end{cases} \\ J_{t+1}(S_t) &\leftarrow J_t(S_t) + \alpha\delta_t\mathbf{z}_t(S_t) \end{aligned} \tag{5.13}$$

For notational purposes, we define the trace parameter for the value and secondary estimators as κ and $\bar{\kappa}$ respectively. Both of these parameters are independent of the λ-return for which we estimate the variance. That is, we are entirely free to estimate the variance of the λ-return for any value of λ independently of the use of any traces in either the value or secondary estimator.

State-Dependent γ. While most RL methods focus on fixed discounting values, it is straightforward to use state-based discounting (Sutton, Modayil, et al., 2011; Modayil, White, et al., 2014), where $\gamma_t \equiv \gamma(S_t)$ (White (2017) goes further by defining transition based discounting). This generalization enables a wider variety of returns to be considered.

First, it allows a convenient means of describing both episodic and continuing tasks and provides an algorithmic mechanism for terminating an episode without defining a recurrent terminal state explicitly. Further, it allows for event-based terminations (Sutton, Modayil, et al., 2011). It also enables soft terminations that may prove useful when training an agent with sub-goals (White, 2017). The use of state-dependent discounting functions is relatively new and remains to be extensively explored.

Off-policy learning. Value estimates are made with respect to a target policy, π. If the behavior policy $\mu = \pi$ then we say that samples are collected on-policy, otherwise, the samples are collected off-policy. An off-policy learning approach is to weight each update by the importance sampling ratio: $\rho_t = \frac{\pi(S_t,A_t)}{\mu(S_t,A_t)}$. There are two different scenarios to be considered when estimating the variance of the return in the off-policy setting. The first is estimating the variance of the on-policy return of the target policy while following a different behavior policy. The second has the goal of estimating the variance of the off-policy return itself. The off-policy λ-return is:

$$G_t^{\lambda:\rho} = \rho_t(C_{t+1} + \gamma_{t+1}(1-\lambda_{t+1})j_t(S_{t+1}) \ + \gamma_{t+1}\lambda_{t+1}G_{t+1}^{\lambda:\rho}). \tag{5.14}$$

where the multiplication by the potentially large importance sampling ratios can significantly increase variance.

It is important to note you would only ever estimate one or the other of these settings with a given estimator. Let η be the weighting for the value estimator, and $\bar{\rho}$ the weighting for the variance estimator. If estimating the variance of the target return from off-policy samples, the first scenario, $\eta_t = 1 \ \forall t$ and $\bar{\rho}_t = \rho_t$. If estimating the variance of the off-policy return $\bar{\rho}_t = 1 \ \forall t$ and $\eta_t = \rho_t$.

Extended Derivation

Theorem 3.

$$v(s) = \mathbb{E}[(\eta_t\delta_t + (\eta_t - 1)j(s))^2 + \lambda_{t+1}^2\gamma_{t+1}^2\eta_t^2 v(S_{t+1})|S_t = s]$$

Proof. The proof is similar to the proof of Theorem 1.

$$\begin{aligned}
v(s) =&\mathbb{E}[\{G_t^\lambda - j(S_t)\}^2|S_t = s] \\
=&\mathbb{E}[\{\eta_t C_{t+1} + \eta_t\gamma_{t+1}(1-\lambda_{t+1})j(S_{t+1}) + \eta_t\gamma_{t+1}\lambda_{t+1}G_{t+1}^\lambda - j(s)\}^2|S_t = s] \\
=&\mathbb{E}[\{\eta_t C_{t+1} + \eta_t\gamma_{t+1}j(S_{t+1}) - \eta_t j(s) + \eta_t j(s) \\
&- \eta_t\gamma_{t+1}\lambda_{t+1}j(S_{t+1}) + \eta_t\gamma_{t+1}\lambda_{t+1}G_{t+1}^\lambda - j(s)\}^2|S_t = s] \\
=&\mathbb{E}[\{(\eta_t\delta_t \ + (\eta_t - 1)j(s)) + \eta_t\gamma_{t+1}\lambda_{t+1}(G_{t+1}^\lambda - j(S_{t+1}))\}^2|S_t = s] \\
=&\mathbb{E}[(\eta_t\delta_t + (\eta_t-1)j(s))^2 + \eta_t^2\gamma_{t+1}^2\lambda_{t+1}^2(G_{t+1}^\lambda - j(S_{t+1}))^2 \\
&+ 2\eta_t\gamma_{t+1}\lambda_{t+1}(\eta_t\delta_t + (\eta_t-1)j(s))(G_{t+1}^\lambda - j(S_{t+1}))|S_t = s] \\
=&\mathbb{E}[(\eta_t\delta_t + (\eta_t-1)j(s))^2 + \eta_t^2\gamma_{t+1}^2\lambda_{t+1}^2(G_{t+1}^\lambda - j(S_{t+1}))^2 + 2\eta_t^2\gamma_{t+1}\lambda_{t+1}\delta_t(G_{t+1}^\lambda - j(S_{t+1})) \\
&+ 2\eta_t\gamma_{t+1}\lambda_{t+1}(\eta_t-1)j(s)(G_{t+1}^\lambda - j(S_{t+1}))|S_t = s] \qquad (5.15)
\end{aligned}$$

Using Lemma 1, with different fixed functions b, we can conclude that the last two terms are zero, giving

$$v(s) =\mathbb{E}[(\eta_t\delta_t + (\eta_t-1)j(s))^2 + \eta_t^2\gamma_{t+1}^2\lambda_{t+1}^2(G_{t+1}^\lambda - j(S_{t+1}))^2|S_t = s]$$

By the law of total expectation

$$\begin{aligned}
v(s) =&\mathbb{E}[(\eta_t\delta_t + (\eta_t-1)j(s))^2 + \mathbb{E}[\eta_t^2\gamma_{t+1}^2\lambda_{t+1}^2(G_{t+1}^\lambda - j(s'))^2|S_{t+1} = s']|S_t = s] \\
=&\mathbb{E}[(\eta_t\delta_t + (\eta_t-1)j(s))^2 + \eta_t^2\gamma_{t+1}^2\lambda_{t+1}^2 v(S_{t+1})|S_t = s].
\end{aligned}$$

completing the proof. □

Theorem 3 gives a Bellman equation for $V(s)$ in the more general off-policy setting. The resulting TD algorithm uses meta-reward $(\eta_t\delta_t + (\eta_t - 1)j(s))^2$ and discounting function $\eta_t^2\gamma_{t+1}^2\lambda^2$.

The Extended Algorithms

To estimate V, our method uses both value and variance estimators. The *value estimator* provides an estimate of the expected return. The *variance estimator*, on the other hand, uses the value estimator to provide an estimate of the variance of the return. Our method, DVTD, and the indirect method, VTD, can be seen as simply defining a meta-reward and a discounting function and can thus be learned with any known TD method, such as TD with

accumulating traces as shown in Equation 5.13. Table 5.1 summarizes our notation.

Direct Variance Algorithm - DVTD

$$\begin{aligned}
\bar{C}_{t+1} &\leftarrow (\eta_t \delta_t + (\eta_t - 1) J_{t+1}(s))^2 \\
\bar{\gamma}_{t+1} &\leftarrow \gamma_{t+1}^2 \lambda_{t+1}^2 \eta_t^2 \\
\bar{\delta}_t &\leftarrow \bar{C}_{t+1} + \bar{\gamma}_{t+1} V_t(s') - V_t(s) \\
\bar{\mathbf{z}}_t(s) &\leftarrow \begin{cases} \bar{\rho}_t(\bar{\gamma}_t \bar{\kappa}_t \bar{\mathbf{z}}_{t-1}(s) + 1) & s = S_t \\ \bar{\rho}_t(\bar{\gamma}_t \bar{\kappa}_t \bar{\mathbf{z}}_{t-1}(s)) & \forall s \in \mathcal{S}, s \neq S_t \end{cases} \\
V_{t+1}(s) &\leftarrow V_t(s) + \bar{\alpha} \bar{\delta}_t \bar{\mathbf{z}}_t(s)
\end{aligned} \tag{5.16}$$

We also present the full VTD algorithm below (again, shown with accumulating traces). Note that this algorithm does not impose that the variance be non-negative.

Second Moment Algorithm - VTD

$$\begin{aligned}
\bar{G}_t &\leftarrow C_{t+1} + \gamma_{t+1}(1 - \lambda_{t+1}) J_{t+1}(s') \\
\bar{C}_{t+1} &\leftarrow \eta_t^2 \bar{G}_t^2 + 2\eta_t^2 \gamma_{t+1} \lambda_{t+1} \bar{G}_t J_{t+1}(s') \\
\bar{\gamma}_{t+1} &\leftarrow \eta_t^2 \gamma_{t+1}^2 \lambda_{t+1}^2 \\
\bar{\delta}_t &\leftarrow \bar{C}_{t+1} + \bar{\gamma}_{t+1} M_t(s') - M_t(s) \\
\bar{\mathbf{z}}_t(s) &\leftarrow \begin{cases} \bar{\rho}_t(\bar{\gamma}_t \bar{\kappa}_t \bar{\mathbf{z}}_{t-1}(s) + 1) & s = S_t \\ \bar{\rho}_t(\bar{\gamma}_t \bar{\kappa}_t \bar{\mathbf{z}}_{t-1}(s)) & \forall s \in \mathcal{S}, s \neq S_t \end{cases} \\
M_{t+1}(s) &\leftarrow M_t(s) + \bar{\alpha} \bar{\delta}_t \bar{\mathbf{z}}_t(s) \\
V_{t+1}(s) &\leftarrow M_{t+1}(s) - J_{t+1}(s)^2
\end{aligned} \tag{5.17}$$

5.4 Experiments

The primary purpose of these experiments is to demonstrate that both algorithms can approximate the true expected V under various conditions in the tabular setting. We consider two domains. The first is an undiscounted, deterministic chain, which is useful for basic evaluation and gives results that are easy to interpret (Figure 5.2). The second is a randomly generated MDP, with different discount and trace-decay parameters in each state (Figure 5.3). For all experiments Algorithm 5.13 is used as the value estimator. Unless otherwise stated, traces are not used ($\kappa = \bar{\kappa} = 0$) and estimates were initialized to zero. For

each experimental setting the average of 30 separate experiments is presented with standard deviation shown as shaded regions. True values were determined by Monte Carlo estimation and are shown as dashed lines in the figures.

We look at the effects of relative step-size between the value estimator and the variance estimators in Section 5.4.1. Then, in Section 5.4.2 we use the random MDP to show that both algorithms can estimate the variance with state-dependent γ and λ. In Section 5.4.3 we evaluate the two algorithms' responses to errors in the value estimate. Section 5.4.4 looks at the effect of using traces in the estimation method. We then examine the off-policy setting in Section 5.4.5. Finally, Section 5.4.6 provides experimental results in a linear function approximation setting.

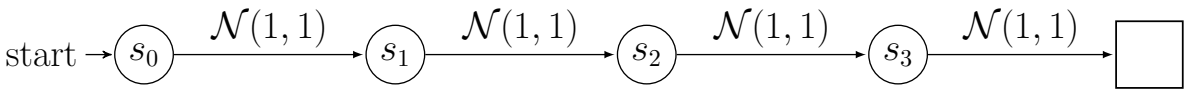


Figure 5.2: **Chain MDP** with 4 non-terminal states and 1 terminal state. From non-terminal states there is a single action with a deterministic transition to the right. On each transition, rewards are drawn from a normal distribution with mean and variance of 1.0. Evaluation was performed for $\lambda = 0.9$, which was chosen because it is not at either extreme and because 0.9 is a commonly used value for many RL experimental domains.

5.4.1 The Effect of Step-Size

We use the chain MDP to investigate the impact of step-size choice. In Figure 5.4a all step-sizes are the same ($\alpha = \bar{\alpha} = 0.001$) and here both algorithms behave similarly. For Figure 5.4b the step-size of the value estimate, ($\alpha = 0.01$), is greater than that of the secondary estimators, ($\bar{\alpha} = 0.001$). Now DVTD smoothly approaches the correct value, while VTD first dips well below zero. This is expected as the estimates are initialized to zero and the variance is calculated as $V(s) = M(s) - J(s)^2$. If the second moment lags behind the value estimate, then the variance will be negative. In Figure 5.4c the step-size for the secondary estimators is larger than for the value estimator ($0.001 = \alpha < \bar{\alpha} = 0.01$). While both methods overshoot the target in this example, VTD has greater overshoot. For both cases of unequal step-size, we see higher variance in the estimates for VTD.

Figure 5.5 explores this further. Here the value estimator is initialized to the true values and updates are turned off ($\alpha = 0$). The secondary estimators are initialized to zero and

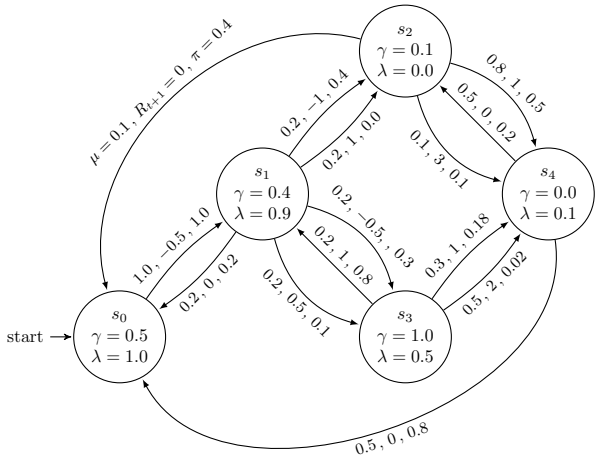
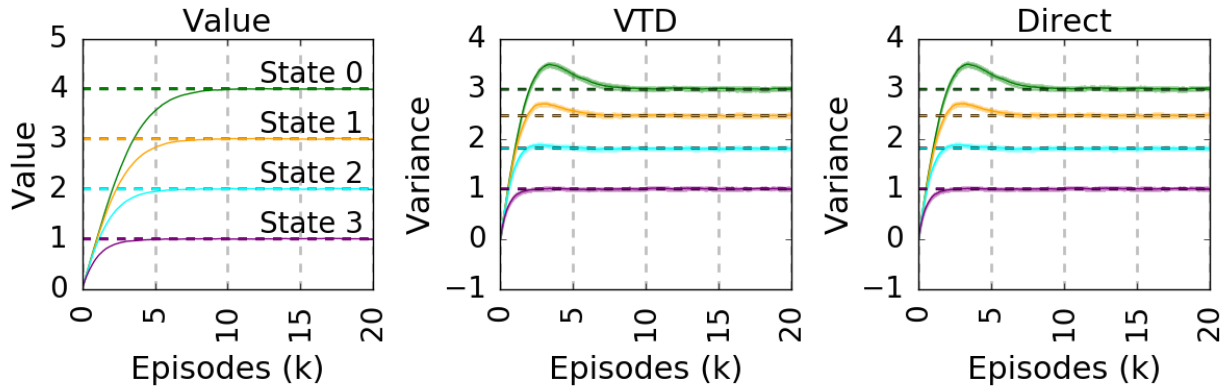


Figure 5.3: **Random MDP**, with a stochastic policy and state-based γ and λ. The state-dependent values of γ and λ are chosen to provide a range of values, with at least one state acting as a terminal state where $\gamma = 0$. On-policy action probabilities are indicated by μ and off-policy ones by π.

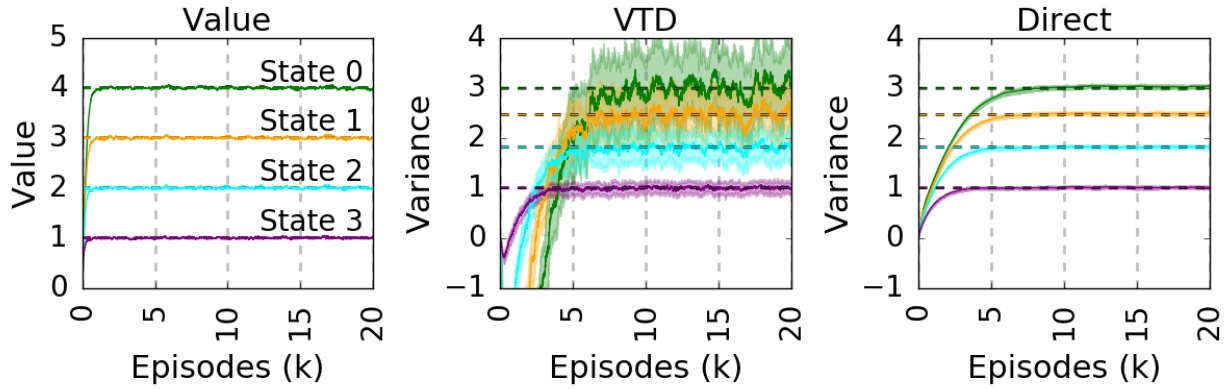
learn with $\bar{\alpha} = 0.001$, chosen simply to match the step-sizes used in the previous experiments. Despite being given the true values the VTD algorithm produces higher variance in its estimates, suggesting that VTD is dependent on the value estimator tracking the return samples.

This sensitivity to step-size ratio is shown in Figure 5.6 where consider different pairs of step-sizes between primary and secondary estimators. All estimates are initialized to their true values. For each ratio, we computed the average variance of the 30 runs of 2000 episodes. We can see that DVTD is largely insensitive to step-size ratio, but that VTD has higher mean squared error (MSE) except when the step-sizes are equal. This result holds for the other experimental settings of this chapter, including the random MDP, but further results are omitted for brevity.

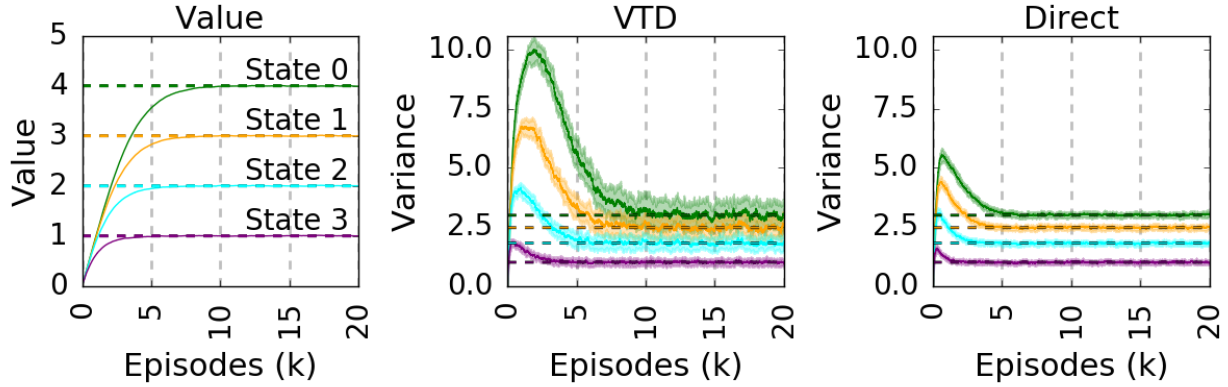
Would there ever be a situation where different step-sizes between value and secondary estimators is justified? The automatic tuning of parameters, such as step-size, is an important area of research, seeking to make learning algorithms more efficient, robust and easier to deploy. Methods that automatically set the step-sizes may produce different values specific to the performance of each estimator. One such algorithm is ADADELTA, which adapts the



(a)



(b)



(c)

Figure 5.4: **Chain MDP** ($\lambda = 0.9$). Varying the ratio of step-size between value and variance estimators. **a)** Step-sizes equal. $\alpha = \bar{\alpha} = 0.001$. **b)** Variance step-size smaller. $\alpha = 0.01, \bar{\alpha} = 0.001$. **c)** Variance step-size larger. $\alpha = 0.001, \bar{\alpha} = 0.01$. We see greater variance in the estimates and greater over/undershoot for VTD when step-sizes are not equal. Shading indicates standard deviation.

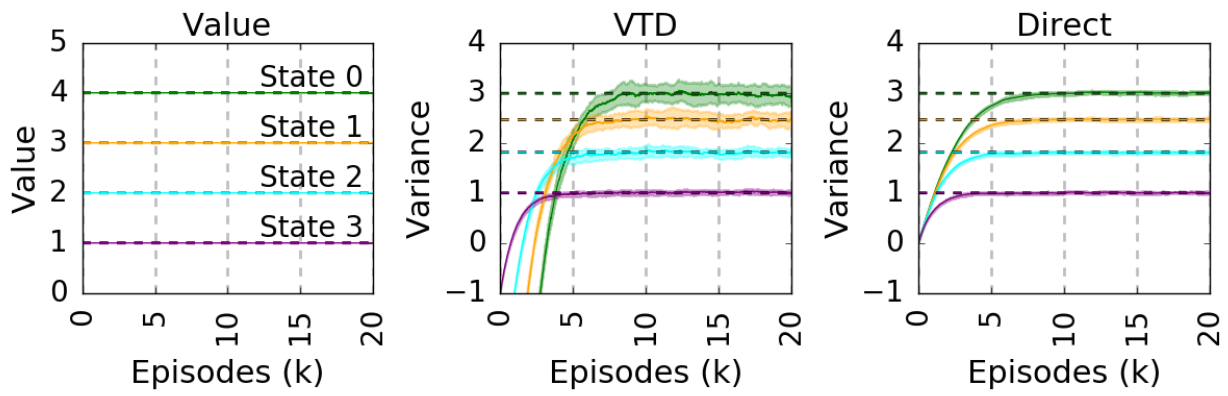


Figure 5.5: **Chain MDP** ($\lambda = 0.9$). Value estimate held fixed at the true values ($\alpha = 0, \bar{\alpha} = 0.001$). Notice the increased estimate variance for VTD, especially State 0.

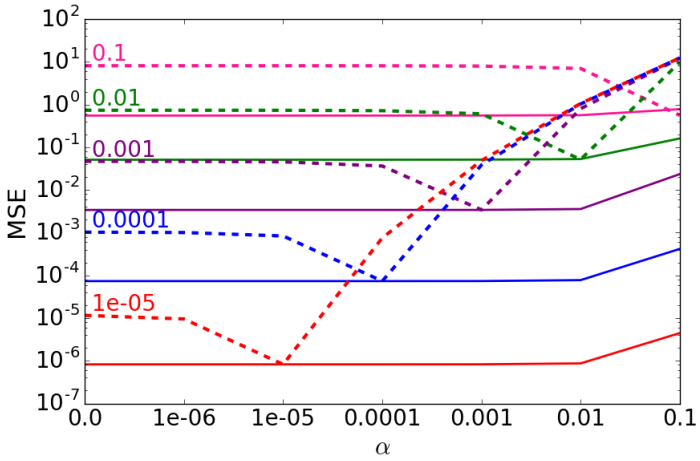
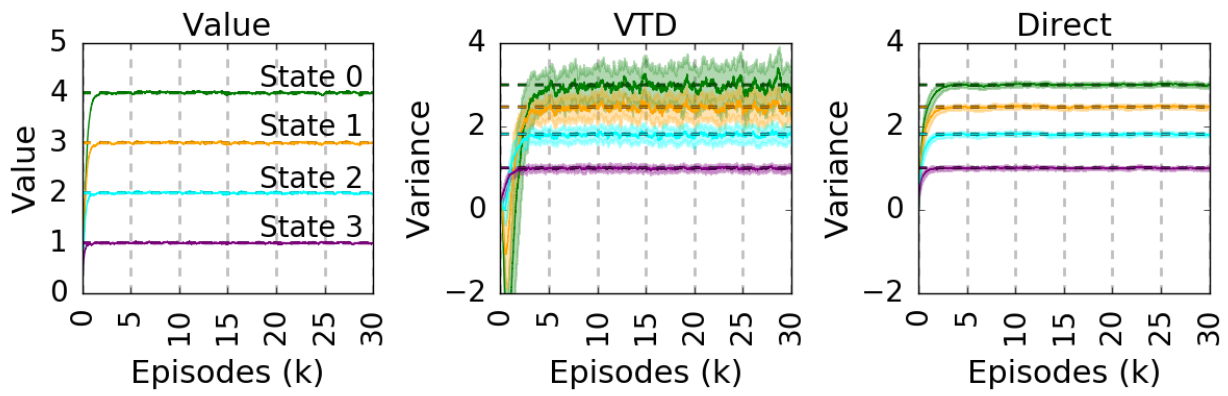
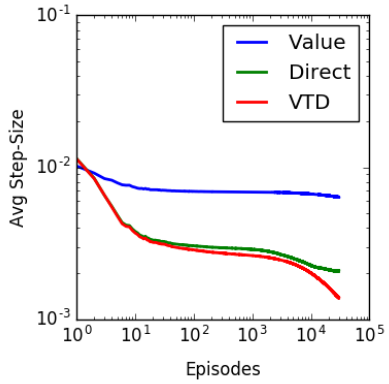


Figure 5.6: **Chain MDP** ($\lambda = 0.9$). The MSE summed over all states as a function of ratios between the value step-size α (shown along the x-axis) and the variance step-size $\bar{\alpha}$ (shown in the 5 series). The direct algorithm is indicated by the solid lines, and VTD is indicated by the dashed. The MSE of the VTD algorithm is higher than the direct algorithm, except when the step-size is the same for all estimators, $\alpha = \bar{\alpha}$ or for very small $\bar{\alpha}$.

step-size based on the TD-error of the estimator (Zeiler, 2012). Figure 5.7 shows that using a separate ADADELTA ($\rho = 0.99, \epsilon = 1e-6$) step-size calculation for each estimator results in higher variance for VTD as expected, given that the value estimator and VTD produce different TD-errors.



(a)



(b)

Figure 5.7: **Chain MDP** ($\lambda = 0.9$). **a)** Results using ADADELTA algorithm to automatically and independently set the step-sizes α and $\bar{\alpha}$. **b)** The step-sizes produced by ADADELTA.

5.4.2 State-dependent γ and λ.

One of the contributions of VTD was the generalization to support state-based γ and λ. Here we evaluate the random MDP from Figure 5.3 (in the on-policy setting, using μ), which was designed for this scenario and which has a stochastic policy, is continuing, and has multiple possible actions from each state. Both algorithms achieved similar results (Figure 5.8).

5.4.3 Variable Error in the Value Estimates

The derivation of our DVTD assumes access to the true value function. The experiments of the previous sections demonstrate that both methods are robust under this assumption, in the sense that the value function was estimated from data and used to estimate V. It

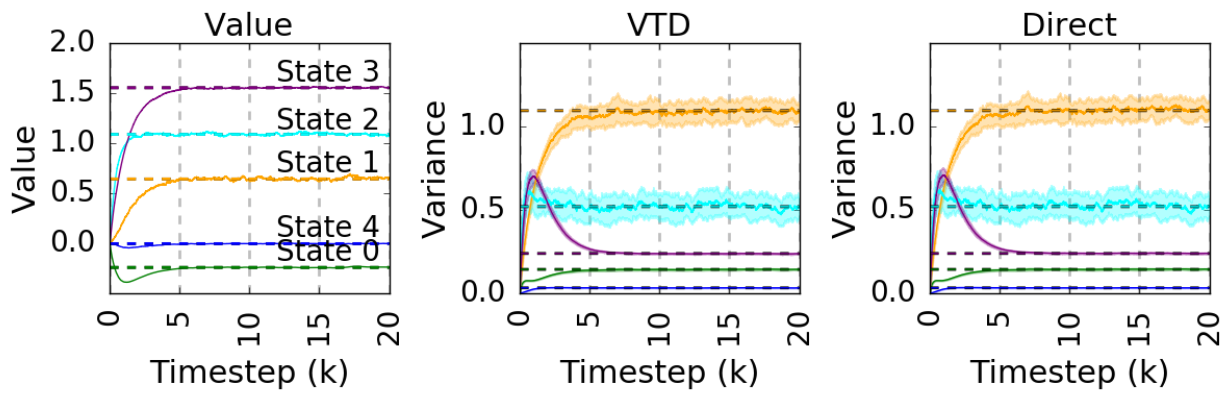


Figure 5.8: **Random MDP** evaluated on-policy with all step-sizes equal ($\alpha = \bar{\alpha} = 0.01$). Both algorithms achieve similar results.

remains unclear, however, how well these methods perform when the value estimates converge to biased solutions.

To examine this, we again use the random MDP shown by Figure 5.3. True values for the value functions and variance estimates are calculated from Monte Carlo simulation of 10,000,000 timesteps. For each run of the experiment each state of the value estimator was initialized to the true value plus an error ($J(s)_0 = j(s) + \epsilon(s)$) drawn from a uniform distribution: $\epsilon(s) \in [-\zeta, \zeta]$, where $\zeta = \max_s(|v(s)|) * \text{err ratio}$ (the maximum value in this domain is 1.55082409). The value estimate was held constant throughout the run ($\alpha = 0.0$). The experiment consisted of 120 runs of 80,000 timesteps. To look at the steady-state response of the algorithms we use only the last 10,000 timesteps in our calculations. Figure 5.9 plots the average variance estimate for each state with the average standard deviation of the estimates as the shaded regions. Sweeps over step-size were conducted, $\bar{\alpha} \in [0.05, 0.04, 0.03, 0.02, 0.01, 0.007, 0.005, 0.003, 0.001]$, and the MSE evaluated for each state. Each data point is for the step-size with the lowest MSE for that error ratio and state. While the average estimate is closer to the true values for VTD, the variance of the estimates is much larger. Further, the average estimates for VTD are either unchanged or move negative, while those of the direct algorithm tend toward positive bias.

For Figure 5.10 the MSE is summed over all states. Again, for each error ratio the MSE was compared over the same step-sizes as before and, for each point, the smallest MSE is plotted. These results suggest the direct algorithm is less affected by error in J.

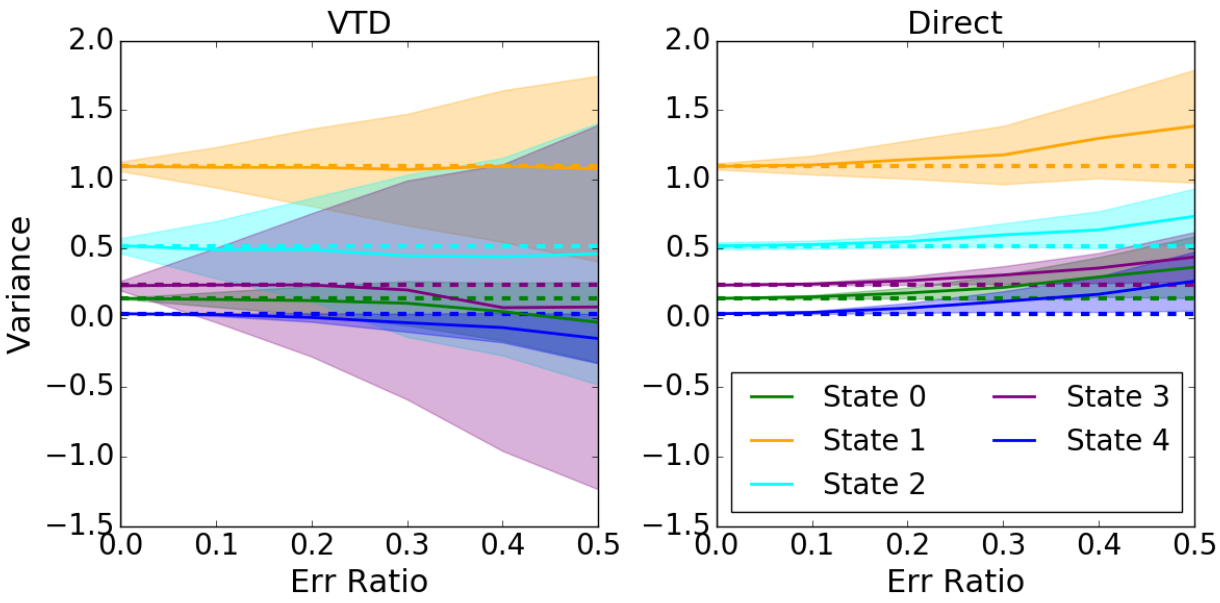


Figure 5.9: **Random MDP**. For each run, the value estimate of each state is offset by a random noise drawn from a uniform distribution whose size is a function of an error ratio and the maximum true value in the MDP. Standard deviation of the estimates is shown by shading.

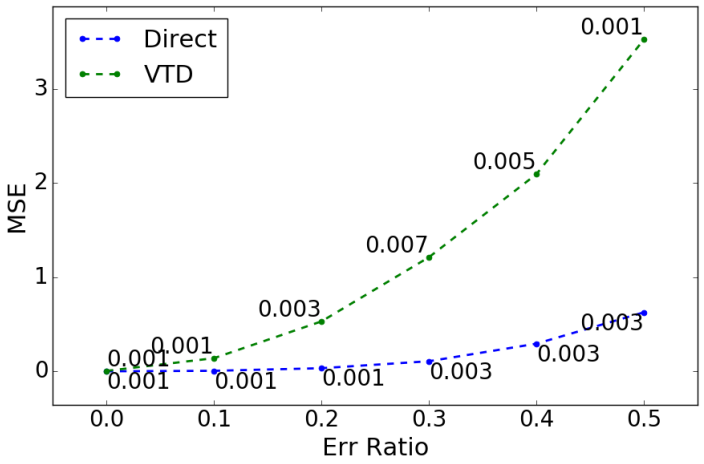


Figure 5.10: **Random MDP**. The MSE computed for the last 10,000 timesteps of 120 runs summed over all states using the step-size with the lowest overall MSE at each error ratio. For each point the step-size, $\bar{\alpha}$, is displayed.

5.4.4 Using Traces

We briefly look at the behavior of the random MDP when traces are used. We found no difference when traces are only used in the secondary estimator and not in the value estimator ($\kappa = 0.0, \bar{\kappa} = 1.0$ (Figure 5.11)

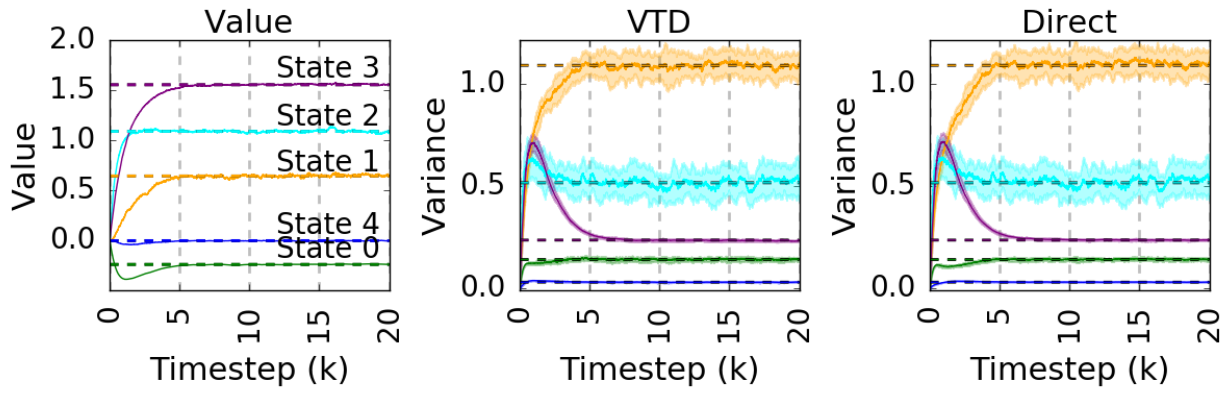


Figure 5.11: **Random MDP**. Using traces in the secondary estimators, but not in the value estimator (TD(λ), $\alpha = \bar{\alpha} = 0.01$, $\kappa = 0.0, \bar{\kappa} = 1.0$). There is no significant difference in performance between DVTD and VTD.

Figure 5.12 considers the opposite scenario, where traces are only used in the value estimator ($\kappa = 1.0, \bar{\kappa} = 0.0$). Here we do see a difference. Particularly the VTD method shows more variance in its estimates for State 0 and 3.

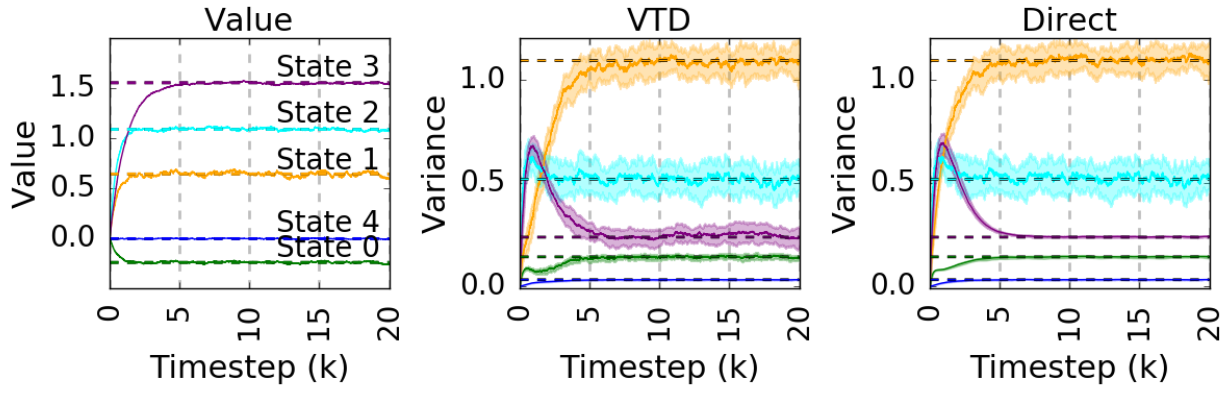


Figure 5.12: **Random MDP**. Using traces (TD(λ), $\alpha = \bar{\alpha} = 0.01$). Traces only used in value estimator ($\kappa = 1.0, \bar{\kappa} = 0.0$). Notice the slight increase in the variance of the VTD estimates for State 0 and 3.

5.4.5 Off-policy Learning

We evaluate two different off-policy scenarios on the random MDP. First, we estimate V under the target policy from off-policy samples. That is, we estimate the V that would be observed if we followed the target policy, i.e., $\eta = 1, \bar{\rho} = \rho$. Both methods achieved similar results in this setting (Figure 5.13).

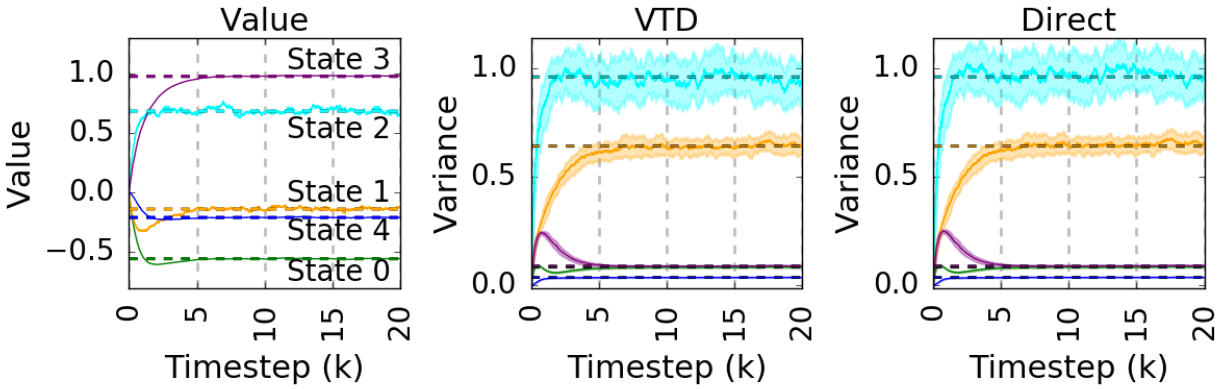


Figure 5.13: **Random MDP** estimating V from off-policy samples ($\alpha = \bar{\alpha} = 0.01, \eta = 1, \bar{\rho} = \rho$). Both methods produce similar results.

In the second off-policy setting, we estimate the variance of the off-policy return (Equation 5.14). Here $\bar{\rho} = 1$ and $\eta = \rho$. Figure 5.14 shows that both algorithms successfully estimate the return in this setting. However, despite having the same step-size as the value estimator, VTD produces higher variance in its estimates, as is most clearly seen in State 3.

5.4.6 Function Approximation

While this chapter has focused on the tabular case, where each state is represented uniquely, here we include a first empirical result in the function approximation setting. We evaluate both methods on the random walk shown in Figure 5.15. This domain was previously used by Tamar, Castro, et al. (2016) for indirectly estimating the variance of the return with LSTD(λ). We use transition based γ (White, 2017) to remove the terminal state and translate the task into a continuing task. Further, we alter the state representation to make it more amenable to TD(λ). For a state s_i we used $\phi_J(i) = [1, (i+1)/30]^T$ as features for the

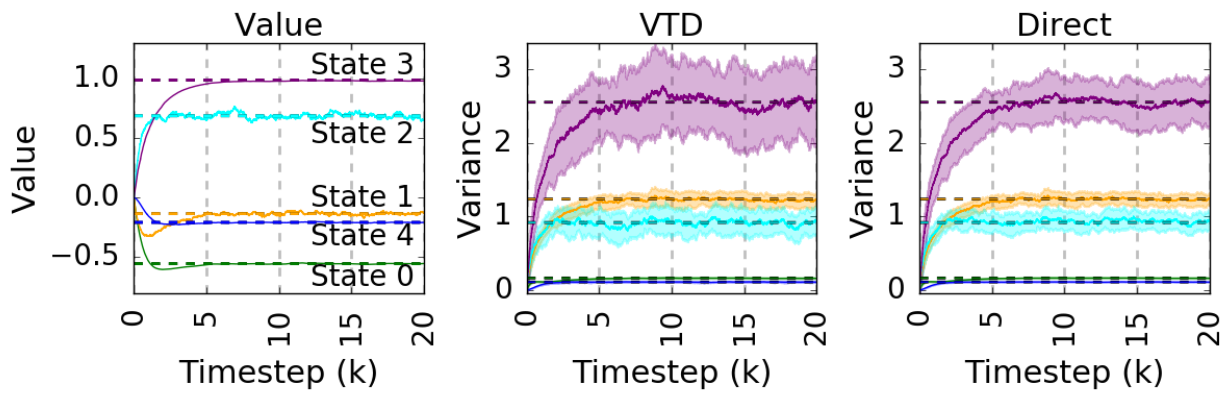


Figure 5.14: **Random MDP** estimating the variance of the off-policy return ($\alpha = \bar{\alpha} = 0.01, \bar{\rho} = 1, \eta = \rho$).

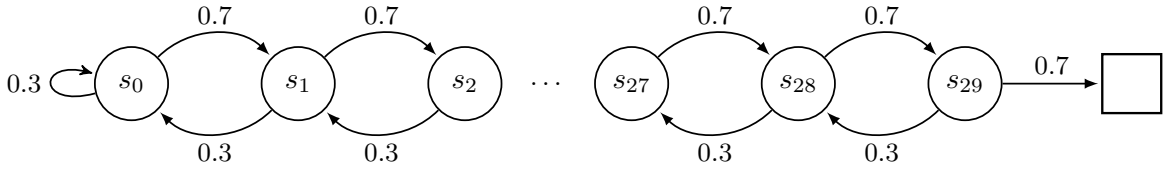


Figure 5.15: **Random Walk**. Random walk with rewards of -1 for every transition to a non-terminal state. Note that there is no discounting in this domain.

value learner and $\phi_M(i) = \phi_V(i) = [1, (i+1)/30, (i+1)^2/30^2]^T$ as features for the secondary learner. We use linear function approximation; each of our estimators is simply an inner product of a set of weights and a feature vector: $J(s) = \mathbf{w}_J^\top \phi_J(s)$, $M(s) = \mathbf{w}_M^\top \phi_M(s)$, and $V(s) = \mathbf{w}_V^\top \phi_V(s)$. DVTD with linear function approximation and accumulating traces is given by modifying Algorithm 5.16:

DVTD with Linear Function Approximation

$$
\begin{aligned}
\bar{C}_{t+1} &\leftarrow (\eta_t \delta_t + (\eta_t - 1) J_{t+1}(s))^2 \\
\bar{\gamma}_{t+1} &\leftarrow \gamma_{t+1}^2 \lambda_{t+1}^2 \eta_t^2 \\
\bar{\delta}_t &\leftarrow \bar{C}_{t+1} + \bar{\gamma}_{t+1} V_t(s') - V_t(s) \\
\bar{\mathbf{z}}_t &\leftarrow \bar{\rho}_t (\bar{\gamma}_t \bar{\kappa}_t \bar{\mathbf{z}}_{t-1} + \phi_t(S_t)) \\
\mathbf{w}_{V:t+1} &\leftarrow \mathbf{w}_{V:t} + \bar{\alpha} \bar{\delta}_t \bar{\mathbf{z}}_t
\end{aligned}
\tag{5.18}
$$

For DVTD, variance is computed directly as $V_{t+1}(s)$. VTD with linear function approximation is given as:

VTD with Linear Function Approximation

$$\begin{aligned}
\bar{G}_t &\leftarrow C_{t+1} + \gamma_{t+1}(1-\lambda_{t+1})J_{t+1}(s') \\
\bar{C}_{t+1} &\leftarrow \eta_t^2 \bar{G}_t^2 + 2\eta_t^2 \gamma_{t+1}\lambda_{t+1}\bar{G}_t J_{t+1}(s') \\
\bar{\gamma}_{t+1} &\leftarrow \eta_t^2 \gamma_{t+1}^2 \lambda_{t+1}^2 \\
\bar{\delta}_t &\leftarrow \bar{C}_{t+1} + \bar{\gamma}_{t+1} M_t(s') - M_t(s) \\
\bar{\mathbf{z}}_t &\leftarrow \bar{\rho}_t(\bar{\gamma}_t \bar{\kappa}_t \bar{\mathbf{z}}_{t-1} + \phi_t(S_t)) \\
\mathbf{w}_{M:t+1}(s) &\leftarrow \mathbf{w}_{M:t}(s) + \bar{\alpha}\bar{\delta}_t \bar{\mathbf{z}}_t(s)
\end{aligned} \tag{5.19}$$

For VTD, variance is computed as $V_{t+1}(s) = M_{t+1}(s) - J_{t+1}(s)^2$.

We define the RMSVE for a value and variance estimate on this domain to be

$$\text{RMSVE}(J) = \sqrt{\sum_{i=0}^{29} d_\pi(s_i)(J(s_i) - j(s_i))^2}$$

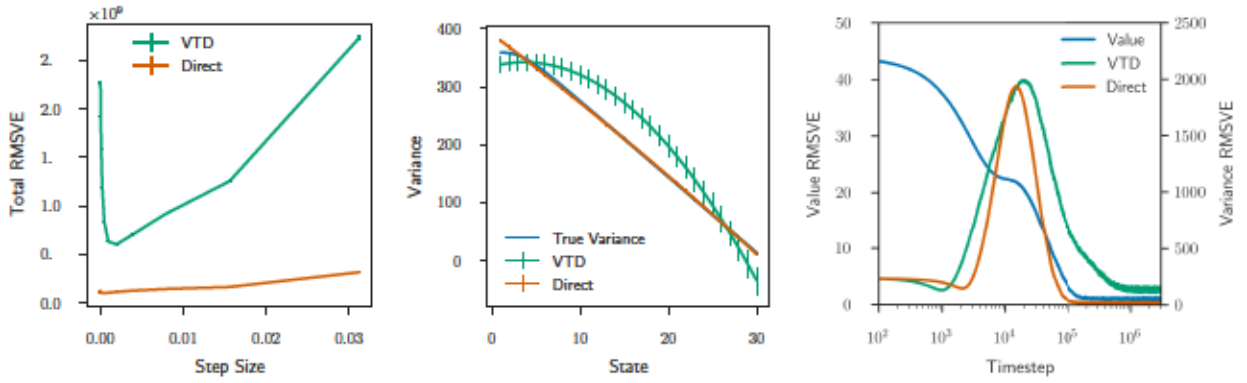
$$\text{RMSVE}(V) = \sqrt{\sum_{i=0}^{29} d_\pi(s_i)(V(s_i) - v(s_i))^2}$$

where $d_\pi(s_i)$ is the steady state probability of being in state s_i. We set $\kappa = \bar{\kappa} = 0.95$ and performed sweeps over step-sizes of $\{2^i, i \in \{-15, -12, \ldots, -1, 0\}\}$. We first found the best step-size for the value learner to be $\alpha_J = 2^{-11}$. Using this step-size for the value learner we then swept over step-sizes for DVTD and VTD. In Figure 5.16a we show the total RMSVE over 3,000,000 timesteps as a function of the step-size for the variance learner. Note that we only show step-sizes that did not lead to numerical errors and each of the values reported is the mean of 30 runs. Here we see that the direct method was considerably more insensitive to the step-size selection than VTD. Using these results we selected the best alpha for VTD ($\alpha_J = 2^{-11}$, $\alpha_M = \alpha_V = 2^{-9}$) and used this for the remaining results.

Figure 5.16b shows the final estimates after 3M timesteps. The values reported are the mean of 100 runs. Here we see that DVTD gives much better estimates than VTD, so much so that the line from DVTD overlaps nearly perfectly with the true variance. VTD shows

significant deviation from the true variance and the estimates have higher standard error across the runs.

Figure 5.16c shows the performance of the methods over time. Here we see DVTD drastically outperforms VTD.



(a) (b) (c)

Figure 5.16: **Random Walk**. **a)** Sensitivity to the step-size for VTD and DVTD. Error bars are shown but almost invisible. **b)** Average final estimates of variance reached after 3M timesteps with 100 runs. We see that DVTD overlaps the true variance closely, with negligible standard error, but VTD shows deviation and noticeable standard error. **c)** Results under linear function approximation over time averaged over 100 runs. Shading indicates standard error (negligible).

5.4.7 Variability in Updates

In this section, we show the effective update to $V_t(s)$ on each timestep for each of the two algorithms in the on-policy setting. For notational clarity let $C = C_{t+1}$, $\alpha = \alpha_t$, $\gamma = \gamma_{t+1}$, $\lambda = \lambda_{t+1}$, $S = S_t$, $S' = S_{t+1}$, and $\delta_t = \delta$.

For the direct algorithm the change is just:

$$\Delta V_t(s) = \bar{\alpha}(\delta^2 + \bar{\gamma} V_t(s') - V_t(s)). \tag{5.20}$$

The updates for the VTD algorithm are much more complicated to compute so we will make some assumptions about the domain to simplify the derivation. First, we compute the change in the second moment and value estimators separately.

We first expand the term δ^2 as follows:

$$\begin{aligned}\delta =& c + \gamma J_t(s') - J_t(s) \\ \delta^2 =& (c + \gamma J_t(s'))^2 - 2(c + \gamma J_t(s'))J_t(s) + J_t(s)^2.\end{aligned}$$

Now we expand the change in the second moment estimate, M. To simplify the expansion, we assume that at each transition the agent moves to a new state, i.e., $s_t \neq s_{t+1} \forall t$ (this is not required for our algorithm, but simplifies the expansions below). This assumption holds for both of the tabular domains examined in this chapter. This allows us to substitute $J_{t+1}(s') = J_t(s')$, which greatly simplifies the updates.

$$\begin{aligned}
\Delta M(s) =& \bar{\alpha}[(c + \gamma J_{t+1}(s'))^2 - \bar{\gamma}^2 J_{t+1}(s')^2 + \bar{\gamma} M_t(s') - M_t(s)] \\
=& \bar{\alpha}[(c + \gamma J_t(s'))^2 - \bar{\gamma}^2 J_t(s')^2 + \bar{\gamma} M_t(s') - M_t(s)] \\
=& \bar{\alpha}[(c + \gamma J_t(s'))^2 - 2(c + \gamma J_t(s'))J_t(s) + J_t(s)^2 + 2(c + \gamma J_t(s'))J_t(s) - J_t(s)^2 \\
& - \bar{\gamma}^2 J_t(s')^2 + \bar{\gamma} M_t(s') - M_t(s)] \\
=& \bar{\alpha}[\delta^2 + 2(c + \gamma J_t(s'))J_t(s) - J_t(s)^2 - \bar{\gamma}^2 J_t(s')^2 + \bar{\gamma} M_t(s') - M_t(s)]
\end{aligned}$$

Notice that from the definition of the TD-error: $c + \gamma J_t(s') = \delta + J_t(s)$.

$$\begin{aligned}
=& \bar{\alpha}[\delta^2 + 2(\delta + J_t(s))J_t(s) - J_t(s)^2 - \bar{\gamma}^2 J_t(s')^2 + \bar{\gamma} M_t(s') - M_t(s)] \\
=& \bar{\alpha}[\delta^2 + 2\delta J_t(s) + J_t(s)^2 - \bar{\gamma}^2 J_t(s')^2 + \bar{\gamma} M_t(s') - M_t(s)] \\
=& \bar{\alpha}[\delta^2 + (\bar{\gamma} M_t(s') - \bar{\gamma} J_t(s')^2) - (M_t(s) - J_t(s)^2) + 2\delta J_t(s) - \bar{\gamma}^2 J_t(s')^2 + \bar{\gamma} J_t(s')^2] \\
=& \bar{\alpha}[\delta^2 + \bar{\gamma} V_t(s') - V_t(s) + 2\delta J_t(s) - \bar{\gamma}^2 J_t(s')^2 + \bar{\gamma} J_t(s')^2] \\
=& \bar{\alpha}[\delta^2 + \bar{\gamma} V_t(s') - V_t(s)] + \bar{\alpha}[2\delta J_t(s) - \bar{\gamma}^2 J_t(s')^2 + \bar{\gamma} J_t(s')^2]
\end{aligned}$$

The first half of this equation is the same as the update for the direct algorithm (5.20). Now we expand the change in the variance update for VTD.

$$\begin{aligned}
\Delta V_t(s) =& (M_{t+1}(s) - J_{t+1}(s)^2) - (M_t(s) - J_t(s)^2) \\
=& \Delta M(s) + J_t(s)^2 - J_{t+1}(s)^2 \\
=& \Delta M(s) + J_t(s)^2 - (\alpha\delta + J_t(s))^2 \\
=& \Delta M(s) + J_t(s)^2 - ((\alpha\delta)^2 + 2\alpha\delta J_t(s) + J_t(s)^2) \\
=& \Delta M(s) - (\alpha\delta)^2 - 2\alpha\delta J_t(s).
\end{aligned}$$

Note that in the case that $\alpha = \bar{\alpha}$ this last term cancels out and we're left with:

$$\Delta V_t(s) = \alpha[\delta^2 + \bar{\gamma} V_t(s') - V_t(s)] + \alpha J_t(s')^2(\bar{\gamma} - \bar{\gamma}^2) - (\alpha\delta)^2.$$

Table 5.2: Average updates for various experiments.

Fig.	Value	2^{nd} Moment	VTD	Direct
5.4a	0.00332	0.0157	0.00415	0.00415
5.4b	0.0322	0.0165	0.143	0.00387
5.4c	0.00332	0.156	0.142	0.0419
5.5	0.0	0.0166	0.0166	0.00381
5.7	0.0212	0.0306	0.0752	0.00884
5.8	0.00362	0.00675	0.00381	0.00385
5.13	0.00362	0.00461	0.00303	0.00307
5.14	0.00362	0.0110	0.0116	0.00838

This suggests that VTD will deviate from the direct method more when: α is larger, $J_t(s')$ is larger, $\bar{\gamma} = 0.5$ and for large values of δ. In general, we expect from this equation that the updates for the VTD will be larger than those of the direct method, suggesting a cause for the higher variance of variance estimates across runs as observed for VTD under a number of scenarios.

We also empirically tested this hypothesis, with Table 5.2 showing the updates for the two algorithms across the tabular experiments. For episodic tasks (chain MDP, Figures 5.4a-5.7) the results show the average total absolute change over all states for a given episode averaged across runs and then averaged across all episodes. For the continuing case (complex MDP, Figures 5.8-5.14) the results are the average absolute change for a timestep averaged over all runs and then averaged across the entire run length. The experiments shaded in gray are those where the two algorithms behaved nearly identically. In this case, we see that the average magnitude of updates is nearly identical. For the other experiments, the VTD algorithm showed higher variance in its variance estimates across runs. For these experiments, we see that the average magnitude of the VTD updates is much larger than for the direct algorithm.

5.5 Discussion

Both DVTD and VTD effectively estimate the variance across a range of settings, but DVTD is simpler and more robust. This simplicity alone makes DVTD preferable. The higher variance in estimates produced by VTD is likely due to the larger target that VTD uses in

its learning updates: $\mathbb{E}[X^2] \geq \mathbb{E}[(X - \mathbb{E}[X])^2]$; we show more explicitly how this affects the updates of VTD in Section 5.4.7. We expect the differences between the two approaches to be most pronounced for domains with larger returns than those demonstrated here. Consider the task of a helicopter hovering formalized as a reinforcement learning task (Ng et al., 2004). In the most well-known variants of this problem the agent receives a massive negative reward for crashing the helicopter (e.g., minus one million). In such problems the magnitude and variance of the return is large. Here, estimating the second moment may not be feasible from a statistical point of view, whereas the target of our direct variance estimate should be better behaved. By focusing on simple MDPs we were able to carefully evaluate the properties of these algorithms while keeping them isolated from additional effects like state-aliasing due to function approximation. Further studies in more complex settings, such as function approximation, are left to future work.

5.6 Conclusion

In this chapter we introduced a simple method for estimating the variance of the λ-return using temporal-difference learning. Our approach is simpler than existing approaches, and appears to work better in practice. We performed an extensive empirical study. Our findings suggest that our new method outperforms VTD when: (1) there is a mismatch in step-size between the value estimator and the variance estimator, (2) traces are used with the value estimator, (3) estimating variances of the off-policy return, or (4) there is error in the value estimate.

This concludes Part I of the dissertation. All of these chapters have been motivated by the conjectures made in Chapter 3, primarily that introspective measures should be included as part of an agent's state representation. The experiments in Chapter 4 empirically investigated several different measures and suggested ways in which they might provide an agent with additional information currently lacking from typical state representations. In Chapter 5 we showed how GVFs could be used to estimate one specific introspective measure—the variance of the return. Research into introspective agents remains an open opportunity for continued development, one which we believe will help enable continual learning agents.

Part II

Answers: Representing GVFs

In the second part of this dissertation we provide several contributions to answering GVF questions. That is, we consider how to algorithmically represent and learn GVFs.

In Chapter 6 we presuppose a mechanism for incrementally constructing a collection of GVFs over time and assume that such a mechanism will identify new prediction targets as it progresses. In this setting we show that if the GVFs are represented using the successor representation (SR) then learning of newly added predictions can be sped up. This is because the SR factors the value estimate into two components, one that captures the environment dynamics and the other that predicts the target signal. The SR allows the reuse of the learned environment dynamics across many different cumulants.

The GVF syntax allows us to specify an infinite set of possible predictive questions. Further, discovering what questions to specify is a difficult process. In Chapter 7 we address these two issues with respect to the timescale parameter. We introduce a new method, Γ-nets, which allows a single GVF network to make predictions for arbitrary timescales. This is accomplished by including timescale as one of the input parameters to the function approximation network. The network then effectively generalizes across timescale. This is one approach to making the learning and representation of multiple timescales tractable.

Chapter 6

Faster GVF Learning with the Successor Representation

We propose using the *Successor Representation* (SR) to accelerate learning in a constructive knowledge system based on *General Value Functions* (GVFs). In real-world settings, like robotics for unstructured and dynamic environments, it is difficult and often impossible to model all meaningful aspects of a system and its environment by hand. Instead, robots must learn and adapt to changes in their environment and task, incrementally constructing models from their own experience. GVFs, taken from the field of reinforcement learning (RL), are a way of modelling the world as predictive questions. One approach to such models proposes a massive network of interconnected and interdependent GVFs, which are incrementally added over time. It is reasonable to expect that new, incrementally added predictions can be learned more swiftly if the learning process leverages knowledge gained from past experience. The SR provides a means of capturing regularities that can be reused across multiple GVFs by separating the dynamics of the world from the prediction targets. As a primary contribution of this work, we show that using the SR can improve sample efficiency and learning speed of GVFs in a continual learning setting where new predictions are incrementally added and learned over time. We analyze our approach in a grid-world and then demonstrate its potential on data from a physical robot arm.

6.1 Introduction

A long standing goal in the pursuit of artificial general intelligence is that of knowledge construction—incrementally modelling and explaining the world and the agent's interaction

with it directly from the agent's own experience (Drescher, 1991). This is particularly important in fields such as continual learning (Ring, 1994) and developmental robotics (Oudeyer, Kaplan, and Hafner, 2007), where we expect agents to be capable of learning, dynamically and incrementally, to interact and succeed in complex environments.

One proposed approach for representing such world models is a collection of *General Value Functions* (GVFs) (Sutton, Modayil, et al., 2011), which models the world as a set of predictive questions each defined by a policy of interest, a target signal, and a timescale (discounting schedule) for accumulating the signal of interest. For example, a GVF on a mobile robot could pose the question "How much current will my wheels consume over the next second if I drive straight forward?" Such predictive models should enable a robot to understand its environment, its body and the consequences of its actions. As these models continually improve so too the robot's ability to act should also improve.

GVF questions are typically answered using temporal-difference (TD) methods (Sutton, 1988) from the field of reinforcement learning (RL) (Sutton and Barto, 2018). A learned GVF approximates the expected future value of a signal of interest, directly representing the relationship between the environment, policy, timescale, and target signal as the output of a single predictive unit.

Nevertheless, despite the success RL algorithms have achieved recently (e.g., (Mnih, Kavukcuoglu, et al., 2015; Silver et al., 2017)), methods for answering multiple predictive questions from a single stream of experience (critical in a robotic setting) are known to exhibit sample inefficiency. In our setting of interest, where multitudes of GVFs are learned in an incremental, sample by sample way, this problem is multiplied. Ultimately, the faster an agent can learn to approximate a new GVF, the better.

This chapter focuses on the problem of model construction. Specifically, we show how one can accelerate learning in a constructive knowledge system based on GVFs by sharing the environment dynamics across the different predictors. This is done with the successor representation (SR) (Dayan, 1993), which allows us to learn the world dynamics under a policy independently of any signal being predicted.

We empirically demonstrate the effectiveness of our approach on both a tabular representation and on a robot arm that uses function approximation. We evaluate our algorithm in the continual learning setting where it is not possible to specify all GVFs *a priori*, but rather GVFs are added incrementally during the course of learning. As a key result, we

show that using a learned SR enables an agent to learn newly added GVFs faster than when learning the same GVFs in the standard fashion without the use of the SR.

6.2 Background

We consider an agent interacting with the environment sequentially. We use the standard notation in the reinforcement learning (RL) literature (Sutton and Barto, 2018), modelling the problem as a Markov Decision Process. Starting from state $S_0 \in \mathcal{S}$, at each timestep the agent chooses an action, $A_t \in \mathcal{A}$, according to the policy distribution $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, and transitions to state $S_{t+1} \in \mathcal{S}$ according to the probability transition $p(\cdot|S_t, A_t)$. For each transition $S_t \xrightarrow{A_t} S_{t+1}$ the agent receives a reward, R_t, from the reward function $R(S_t, A_t, S_{t+1}) \in \mathbb{R}$.

In this chapter we focus on the prediction problem in RL, in which the agent's goal is to predict the value of a signal from its current state (e.g., the cumulative sum of future rewards). Note that throughout this chapter we use upper case letters to indicate random variables.

6.2.1 General Value Functions (GVFs)

The most common prediction made in RL is about the expected return. The return is defined to be the sum of future discounted rewards under policy π starting from state s. Formally, $G_t = \sum_{t=0}^{T} \gamma^t R_t$, with $\gamma \in [0, 1]$[1] being the discount factor and T being the final timestep, where $T = \infty$ denotes a continuing task. The function encoding the prediction about the return is known as the value function $v_\pi(s) = \mathbb{E}_\pi[G_t|S_0 = s]$.

GVFs (Sutton, Modayil, et al., 2011) extend the notion of predictions to different signals in the environment. This is done by replacing the reward signal, R_t, by any other target signal, which we refer to as the cumulant, C_t, and by allowing a state-dependent discounting function, $\gamma_t = \gamma(S_t)$, instead of using a fixed discounting factor. The general value of state s under policy π is defined as:

$$v_{\pi:\gamma}(s) = \bar{c}_\pi + \bar{\gamma}_\pi \sum_{s'} p_\pi(s'|s) v_\pi(s'), \tag{6.1}$$

[1]Note that $\gamma = 1$ is only valid when termination is guaranteed as in the episodic case.

where $\bar{c}_\pi$ is the average cumulant from state s, $\bar{\gamma}_\pi$ is the average γ from state s, and $p_\pi(s'|s)$ is the probability of transitioning from state s to s' under policy π. This can also be written in matrix form: $\mathbf{v}_{\pi:\gamma} = \bar{\mathbf{c}}_\pi + \bar{\boldsymbol{\gamma}}_\pi \odot P_\pi \mathbf{v}_{\pi:\gamma}$, where $\odot$ denotes element-wise multiplication. Such an equation, when solved, gives us

$$\mathbf{v}_{\pi:\gamma} = (I - \bar{\boldsymbol{\gamma}}_\pi \odot P_\pi)^{-1} \bar{\mathbf{c}}_\pi, \tag{6.2}$$

where I is the identity matrix, $\mathbf{v}, \bar{\mathbf{c}}, \bar{\boldsymbol{\gamma}} \in \mathbb{R}^{|\mathcal{S}|\times 1}$, and $P_\pi \in [0,1]^{|\mathcal{S}|\times|\mathcal{S}|}$ is a probability matrix such that $[P_\pi]_{ij} = \mathbb{E}[p_\pi(S_{t+1} = s_j | S_t = s_i)]$.

6.2.2 The Successor Representation (SR)

The successor representation (Dayan, 1993) was initially proposed as a representation capable of capturing state similarity in terms of time. It is formally defined, for a fixed $\gamma < 1$, as:

$$\psi_\pi(s, s') = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}_{\{S_t = s'\}} | S_0 = s\Big].$$

In words, the SR encodes the expected number of times the agent will visit a particular state when the sum is discounted by γ over time[2]. It can be re-written in matrix form as

$$\Psi_\pi = \sum_{t=0}^{\infty} (\gamma P_\pi)^t = (I - \gamma P_\pi)^{-1}. \tag{6.3}$$

Importantly, the SR can be easily computed, incrementally, by standard RL algorithms such as TD learning (Sutton, 1988), since its primary modification is to replace the reward signal by a state visitation counter. Nevertheless, despite its simplicity, the SR holds important properties we leverage in this chapter.

The SR, in the limit, for a constant γ (see Eq. 6.3), corresponds to the first factor of the solution in Eq. 6.2. Thus, the SR can be seen as encoding the dynamics of the Markov chain induced by the policy π and by the environment's transition probability p. If the agent has access to the SR, it can accurately predict the (discounted) accumulated value of any signal, from any state, by simply learning the expected *immediate* value, $\bar{\mathbf{c}}_\pi$, of that signal in each state. On the other hand, if the agent does not use the SR, the agent must also deal with the problem of credit assignment, having to look at n-step returns to control for *delayed*

[2]Note that Dayan describes the SR as predicting future state visitation from time t onward. This is non-standard in RL as we typically describe the return as predicting the signal from $t+1$ onward.

consequences. Importantly, the dynamics encoded by the SR are the same for all signals learned under the same policy and discount function. This factorization of the solution is the main property we use in our work, described in the next section.

6.3 Methods

As aforementioned, we are interested in the problem of knowledge acquisition in the continual learning setting (Ring, 1994), where knowledge is encoded as predictive questions (GVFs). In this setting it is not possible to specify all GVFs ahead of time. Instead, GVFs must be added incrementally by some, as yet unknown, mechanism. The standard approach would be to learn each newly added prediction from scratch. In this section we discuss how we can use the SR to accelerate learning by taking advantage of the factorization shown in Eq. 6.2. Our method leverages the fact that the SR is independent of the target signal being predicted, learning the SR separately and re-using it when learning to predict new signals.

In the previous section, for clarity, we discussed the main concepts in the tabular case. In real world applications, where the state space is too large, assuming states can be uniquely identified is not often feasible. Instead, we generally represent states as a set of features $\phi(s) \in \mathbb{R}^d$ where $d \ll |\mathcal{S}|$. Because both ψ and $\bar{c}$ are a function of feature vector $\phi(S)$, they can easily be represented using function approximation and learned using TD algorithms. In order to present a more general version of our algorithm, we introduce it here using the function approximation notation.

The first step in our algorithm is to compute the one-step average cumulant, which we do with TD-error:

$$\delta_t = C_{t+1} - \bar{c}\big(\phi(S_t)\big). \tag{6.4}$$

If we use linear function approximation to estimate $\bar{c}$ then $\bar{c}\big(\phi(s)\big) = \phi(s)^\top \mathbf{w}$. The TD-error for ψ is given as (note that $\boldsymbol{\delta}$ is a vector of length d)

$$\boldsymbol{\delta}_t = \phi(S_t) + \gamma_{t+1}\psi\big(\phi(S_{t+1})\big) - \psi\big(\phi(S_t)\big). \tag{6.5}$$

This generalization of the SR to the function approximation case is known as successor features (Barreto, Borsa, et al., 2018).

Algorithm 1 GVF prediction with the SR

Input: Feature representation ϕ, policy π, discount function γ, and step-sizes α_C, α_{SR}
Output: Matrix M and vectors $\bar{c}_i$ as predictors of C_i

Initialize $\mathbf{w}$ and M arbitrarily
while S' is not terminal **do**
  Observe state S, take action A selected according to $\pi(S)$, and observe a next state S' and the cumulants C_i
  $\delta_{SR} = \phi(S) + \gamma(S')M^\top\phi(S') - M^\top\phi(S)$
  $M \leftarrow M + \alpha_{SR}\phi(S) \otimes \delta_{SR}$
  for each cumulant C_i **do**
    $\delta_{C_i} = C_i - \phi(S)^\top \mathbf{w}_i$
    $\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha_{C_i}\phi(S)\delta_{C_i}$
  end for
end while
$\delta_{SR} = \phi(S') - M^\top\phi(S')$
$M \leftarrow M + \alpha_{SR}\phi(S') \otimes \delta_{SR}$

If we use linear function approximation to estimate ψ then $\psi\big(\phi(s)\big) = M^\top\phi(s)$, where $M \in \mathbb{R}^{d\times d}$. Using the usual semi-gradient method,[3] often used with TD, we derive a TD(0) stochastic gradient descent update as:

$$\begin{aligned} M_{t+1} &= M_t - \frac{1}{2}\alpha\nabla_M(\boldsymbol{\delta}_t^2) \\ &= M_t + \alpha\nabla_M\big(\psi(S_t)\big) \otimes \boldsymbol{\delta}_t \\ &= M_t + \alpha\phi(S_t) \otimes \boldsymbol{\delta}_t, \end{aligned}$$

where ∇_M is the gradient with respect to M and $\otimes$ is the outer product. Based on this derivation, as well as Eq. 6.4 and 6.5, we obtain Algorithm 1. Note that the last two lines of Algorithm 1, which update the SR for the state S', are only required for the episodic case.

This algorithm allows us to predict the cumulant C_i, in state S, using the current estimate of the matrix M and the weights $\mathbf{w}_i$. We can then obtain the final prediction by simply computing $\psi\big(\phi(S)\big)^\top\mathbf{w} = \big(M^\top\phi(s)\big)^\top\mathbf{w} = \phi(s)^\top M\mathbf{w}$.

This algorithm accelerates learning because, generally, learning to estimate $\bar{c}$ is faster than learning the GVF directly. This is exactly what our algorithm does. When predicting a new signal, it starts with its current estimate for the SR, Ψ. At the end, the multiplication

[3]In semi-gradient methods, the effect of M on the prediction target $\phi(S_{t+1}) + \gamma\psi\big(\phi(S_{t+1})\big)$ is ignored when computing the gradient.

$\Psi\bar{c}$ is simply a weighted average of the one-step predictions across all states, weighted by the likelihood they will be visited. We provide empirical evidence supporting this claim in the next sections.

6.4 Evaluation in Dayan's Grid World

We first evaluated our algorithm in a tabular grid world whose simplicity allowed us to analyze our method more thoroughly since we were not bounded by the speed and complexity of physical robots. The grid world we used was inspired by Dayan (1993) (see Figure 6.1). Four actions are available in this environment: up, down, left, right. Taking an action into a wall (blue) results in no change in position. Transitions are deterministic, i.e., a move in any direction moves the agent to the next cell in the given direction, except when moving into a wall. For each episode the agent spawns at location **S** and the episode terminates when the agent reaches the goal **G**.

We generated fifty different signals for the agent to predict. They were generated randomly from a collection of primitives enumerated in Table 6.1. They are composed of two different primitive signals, sig_x and sig_y, one for each axis, like so: $(\text{sig}_x(x + \text{offset}_x) + \text{bias}_x) * (\text{sig}_y(y + \text{offset}_y) + \text{bias}_y)$. The bias and offset were drawn from $[-2.0, 2.0)$ and $[0, 10)$, respectively. Offset and bias were not applied to either the unit or shortest path primitives. Further, the shortest path primitive was not combined with a second signal, but was used on its own. Gaussian noise with a standard deviation of 0.3 was applied on top of each signal. The shortest path signal is inspired by a common reward function used in RL where each transition has a cost (a negative reward) meant to push the agent to completing a task in a timely manner and reaching the goal produces a positive reward signal.

Our agent selects actions using ϵ-greedy action selection where, at each timestep, with probability $1 - \epsilon$, it uses the action specified by a hand-coded policy (see Figure 6.1), and otherwise chooses randomly from all four actions ($\epsilon = 0.3$ in our experiments). A tabular representation is used with each grid cell uniquely represented by a one-hot encoding.

In this set of experiments we can compute the ground-truth predictors for the SR and the signal predictors. This is done by taking the average return observed from each state. The SR reference was averaged over 30,000 episodes and the signal predictor references were averaged over 10,000 episodes. Each episode started at the start state and followed the

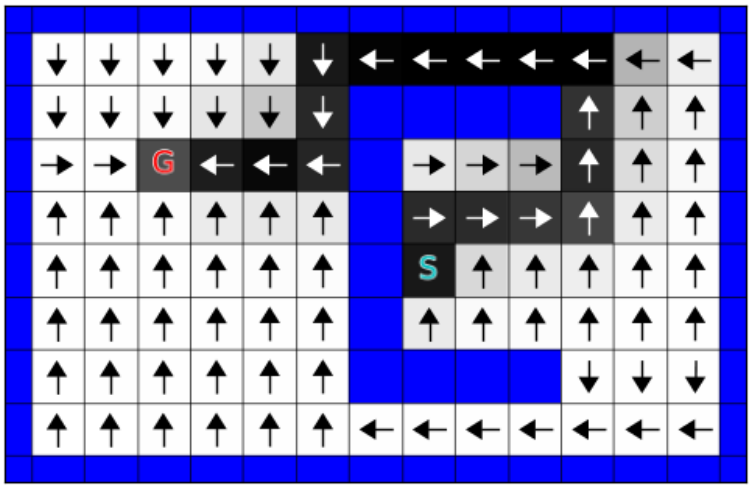
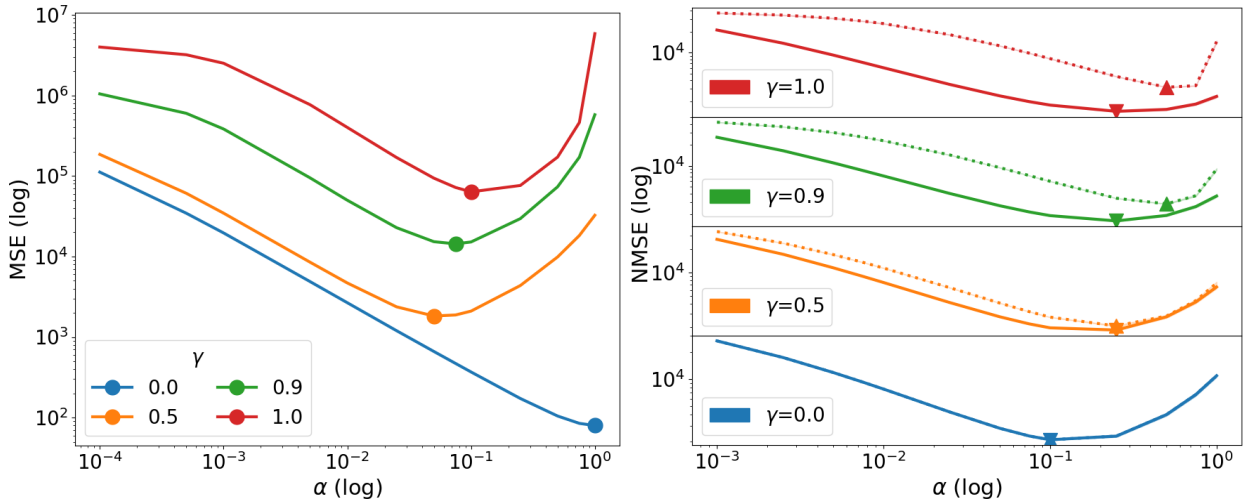


Figure 6.1: Dayan's grid world. Arrows indicate the hand-coded policy (from the start state the policy is to go up). Black squares indicate the SR prediction given from the starting state, S, for $\gamma = 1.0$; the darker the square the higher the expected visitation. Notice the graying around the central path caused by the ϵ-greedy action selection.



(a) (b)

Figure 6.2: **a)** MSE of the SR as a function of step-size α for different values of γ. Lowest error is indicated by the markers. **b)** A comparison of the NMSE of the direct (dashed lines) and SR-based (solid lines) predictions as a function of fixed step-size α and discount factor γ summed across all signals. Lowest cumulative error is indicated by up arrows (direct predictions) and down arrows (SR-based predictions). Note that, although difficult to see, confidence intervals of 95% are included in figures.

ϵ-greedy policy already described.

We first evaluated the predictive performance of our SR learning algorithm with respect to

Table 6.1: Signal Primitives

Primitive	**Parameters**
Fixed value	value $\in [-2.0, 2.0)$
Square wave	period $\in [2, 40)$, invert $\in [True, False]$
Sin wave	period $\in [2, 40)$
Random binary	Fixed random binary string generated over the length of an axis.
Random float	Fixed random floats $\in [0, 1.0)$ generated over the length of an axis.
Unit	Fixed value of 1.0
Shortest path	transition_cost $\in [-10.0, -1.0)$, goal_reward $\in [1.0, 10.0)$

the step-size for a variety of γ values. We report the average over 30 trials of 10,000 episodes. We initialize the SR weights to 0.0. The squared Euclidean distance was calculated between the predicted SR and the reference SR for each timestep. These values were summed over the run and the average was taken across the runs. These averages are shown in Figure 6.2a.

Using the results in Figure 6.2a we evaluated the performance of the two signal prediction approaches by sweeping across step-sizes for various γ. For each experimental run, learning of a new signal was enabled incrementally every 50 episodes. This produced runs with a total length of 2,500 episodes where the first pair of GVFs added (the direct and one-step predictors) were trained for 2,500 episodes and the last added GVFs were trained for 50 episodes. Further, for each run, the order in which the signals were added was randomized. Thirty runs were performed. The weights of the predictors and of the SR were initialized to 0.0. Notice that the SR was being learned at the same time as the direct and one-step predictions.

For each run a cumulative MSE for each signal i was calculated according to Eq. 6.6. This equation computes the total squared error between the predictor's estimate, V and the reference predictor's estimate, V^*. For each episode, E, the error of the current and previous episodes is averaged. Then, for each signal, the maximum error for a given γ, either in the direct or SR-based predictions, is found and used to normalize the errors in the signal across that particular value of γ (see Eq. 6.7). In this way we attempt to treat the error of each signal equally. If this is not done the errors of large magnitude signals dominate the results. These normalized values are then summed across the signals and the averages across all 30 runs are plotted in Figure 6.2b.

$$MSE_i = \frac{1}{E}\sum_{e_0}^{E}\frac{1}{T}\sum_{t=0}^{T}(V_{i:t} - V^*_{i:t})^2 \tag{6.6}$$

$$NMSE_{i:\alpha:\gamma} = \frac{MSE_{i:\alpha:\gamma}}{\max_\alpha(MSE_{i:\gamma:direct}, MSE_{i:\gamma:SR})} \tag{6.7}$$

The advantage of the SR-based method is clear as γ increases. This is to be expected since for $\gamma = 0.0$ both methods are making one-step predictions. In the experiment of Figure 6.2b, the SR performs better in the vast majority of the signals, as shown in Table 6.2. For all step-sizes not listed the SR-based method was better on all signals. Analysis of these cases where the direct method did better reveal that some of the target signals have very small magnitudes, suggesting the SR-based approach may be more susceptible to signal-to-noise ratio.

Table 6.2: Signal performance for $\gamma = 0.9$ of Figure 6.2b.

α	**Direct Better**	**SR-Based Better**
0.25	3	47
0.5	5	45
0.75	4	46
1.0	1	49

Finally, we analyzed how the prediction error of our systems evolved with time. This is demonstrated in Figure 6.3 where we selected the best step-sizes for $\gamma = 0.9$ and plotted the performance over time across 30 different runs. In this case the order of the signals remained fixed so that sensible averages could be plotted for each signal. Signal performance was normalized as before and summed across all active signals. As expected, we clearly see that the SR-based predictions (green) start with much higher error than the direct (blue), but as the error of the SR (red) drops low the newly added SR-based predictors are able to learn quicker, with less peak and overall error than the direct predictor.

In the continual learning setting we never have the opportunity to tune for optimal step-sizes as we did in our evaluation. Practically, fixed step-sizes are used for many robotics settings in RL, but, in order to ensure stable learning, small step-sizes are chosen. As we saw in Figure 6.2b, the advantage of using the SR-based predictions is enhanced with smaller

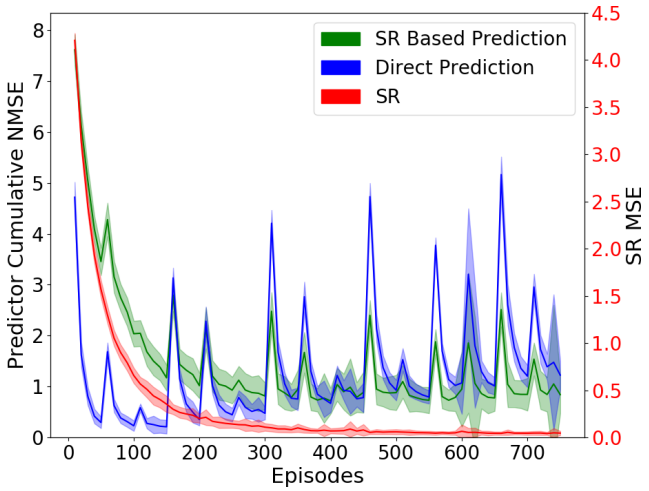


Figure 6.3: All predictors learn from scratch with new predictors added in every 50 episodes. As the SR error (red, right axis) goes low the SR-based predictors (green) are able to learn faster than their direct (blue) counterparts. Shading indicates a 95% confidence interval.

step-sizes. Ideally, however, we would imagine that a fully developed system would use some method of adapting step-sizes, such as ADADELTA (Zeiler, 2012).

6.5 Evaluation on a Robot Arm

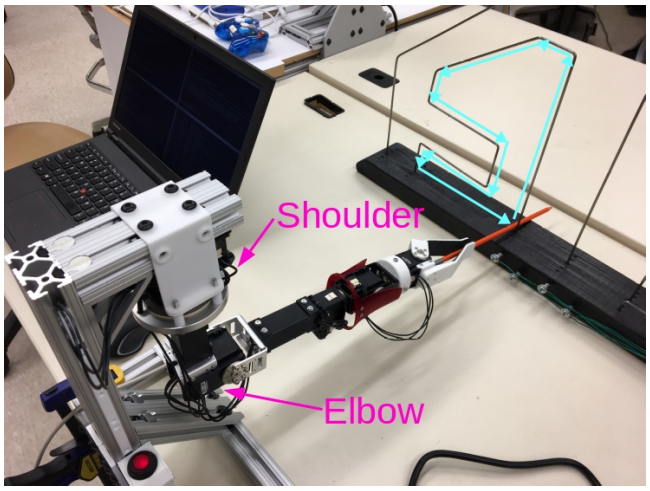


Figure 6.4: The user controls the robot arm using a joystick to trace the inside of the wire maze in a counter-clockwise direction. Circuit path shown in blue.

Tabular settings like Dayan's grid world are useful for enabling analysis and providing

insight into the behavior of our method. However, our goal is to accelerate learning on a real robot where states are not fully observed and cannot be represented exactly; instead we must use function approximation. Here we demonstrate our approach using a robot arm and learning sensorimotor predictions with respect to a human-generated policy. In our task a user controls a robot arm via joystick to trace a counter-clockwise circuit through the inside of the wire maze (see Figure 6.4) with a rod held in the robot's gripper. The user performed this task for approximately 12 minutes completing around 50 circuits.

In this experiment we used six different prediction targets: the current, position and speed of both the shoulder rotation and elbow flexion joints. A new predictor was activated every 2,000 timesteps (Note that the robot reports sensor updates at 30 timesteps/s). For this demonstration a discount factor of $\gamma = 0.95$ was used. Four signals were used as input to our function approximator: the current position and a decaying trace of the position for the shoulder and elbow joints. The decaying trace for joint j was calculated as $tr_{j:t+1} = 0.8 * tr_{j:t} + 0.2 * pos_{j:t+1}$. These inputs were normalized over the joint ranges observed in the experiment and passed into a 4-dimensional tilecoding (Sutton and Barto, 2018) with 100 tilings (each randomly shifted) of width 1.0 and a total memory size of 2048. Additionally a bias unit was added, resulting in a binary feature vector of length 2049 with a maximum of 101 active features on each timestep (hashing collisions can reduce this number). We use a decaying step-size for all the predictors where the step-size starts at 0.1 and decays linearly to zero over the entire dataset. At each timestep this step-size is further divided by the number of active features in $\phi(S_t)$. Finally, for each predictor, this step-size is offset such that the step-size starts at 0.1 when it is first activated and decays at the same rate as all the other predictors.

To compare the prediction error we compute a running MSE for each signal according to Eq. 6.8, where at each timestep t the sum is taken over all previous timesteps. Unlike the previous tabular domain, we do not have the ideal estimator to compare against and instead compare the predictions, V, against the actual return, G. In order to treat each signal equally we further normalize these errors according to Eq. 6.9. Note that the NMSE allows us to compare the predictions of a single signal between the two methods, but does

not tell us how accurate the predictions are nor does it allow comparison across signals.

$$MSE_{i:t} = \frac{1}{t}\sum_{k=1}^{t}(V_{i:t} - G_{i:t})^2 \tag{6.8}$$

$$NMSE_i = \frac{MSE_i}{\max(MSE_{i:direct}, MSE_{i:SR})} \tag{6.9}$$

Figures 6.5 and 6.6 show a single run, approximately 12 minutes in length. A single ordering of the predictors was used. Figure 6.5 shows the error across all predictors while Figure 6.6 separates out each predictor. Here we see a clear advantage to using the SR-based predictions for most of the signals. Unlike the previous tabular results, there is little difference on the performance of the first predictor (shoulder current), even while the SR is being learned. To investigate, we ran experiments where each signal was learned from the beginning of the run. We observed that performance was rarely worse and sometimes even better when using the SR-based method. This suggests the SR-based approach is more robust than expected, but further experimentation is needed.

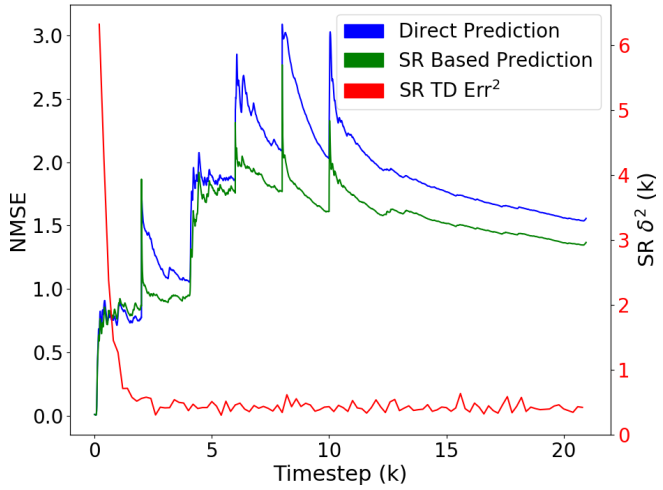


Figure 6.5: Prediction accuracy for a single 12 minute run tracing the maze circuit. A new predictor is added every 2,000 timesteps. NMSE errors are summed across all predictors.

6.6 Further Advantages When Scaling

While this chapter analyzed single policies and discount functions, this is not the setting in which the GVF framework is proposed to be used. Rather, it is imagined that massive

numbers of GVFs over many policies and timescales will be used represent complex models of the world (Sutton, Modayil, et al., 2011; Modayil, White, et al., 2014). In this setting we note that using SR-based predictions can offer additional benefits, allowing the robot to do more with less. Consider, for a single policy π, a collection of SRs learned for f discount functions and h one-step predictors. We can then represent fh predictions using $f + h$ predictors. A first advantage is that far fewer GVFs need to be updated on each timestep, saving computational costs. As a second benefit, there is potential to reduce the number of weights used by the system. For example, consider learning in a tabular setting, with $|\mathcal{S}|$ states, using linear estimators. For fh predictions the number of weights needed is : $|w|_{direct} = fh|\mathcal{S}|$, $|w|_{SR-based} = f|\mathcal{S}|^2 + h|\mathcal{S}|$. It can be shown that for a fixed f and $\mathcal{S}$ the total number of weights used by the direct prediction approach is greater when $h > \frac{f|\mathcal{S}|}{f-1}$.

6.7 Related Work

The SR was originally introduced as a representation method for policy learning (Dayan, 1993), but has recently been applied to other settings. It has been used, for instance, in transfer learning problems allowing agents to generalize better across different tasks (Barreto, Borsa, et al., 2018; Kulkarni et al., 2016; Lehnert et al., 2017; Yao et al., 2014; Zhang et al., 2017; Zhu et al., 2017; Ma et al., 2018). Particularly, the work of Zhang et al. (2017) showed how the SR could be used for transfer in robot navigation tasks. The robot first learned to navigate a simulated 3D maze environment. Next, the robot was moved to a real maze and showed transfer by learning the real-world navigation task faster than learning from scratch. Zhu et al. (2017) demonstrated the use of the SR in transfer with a traditional STRIPS style planning algorithm. In their setting a simulated robot planned over high-level actions with camera images as input. Their approach showed better ability to complete previously unseen tasks than several other well-known algorithms.

To learn optimal policies an RL agent must sufficiently explore its state space. In complex, real-world scenarios, such as those faced by robots, the state-action space is incredibly large and exploration by random primitive actions, as is often done in simple grid-world domains, is not feasible. The work of Machado, Rosenbaum, et al. (2018) uses the SR to define intrinsic rewards in an option discovery algorithm. By exploring with these options agents are able to improve the efficiency of their exploration. Such an approach might be applied beneficially

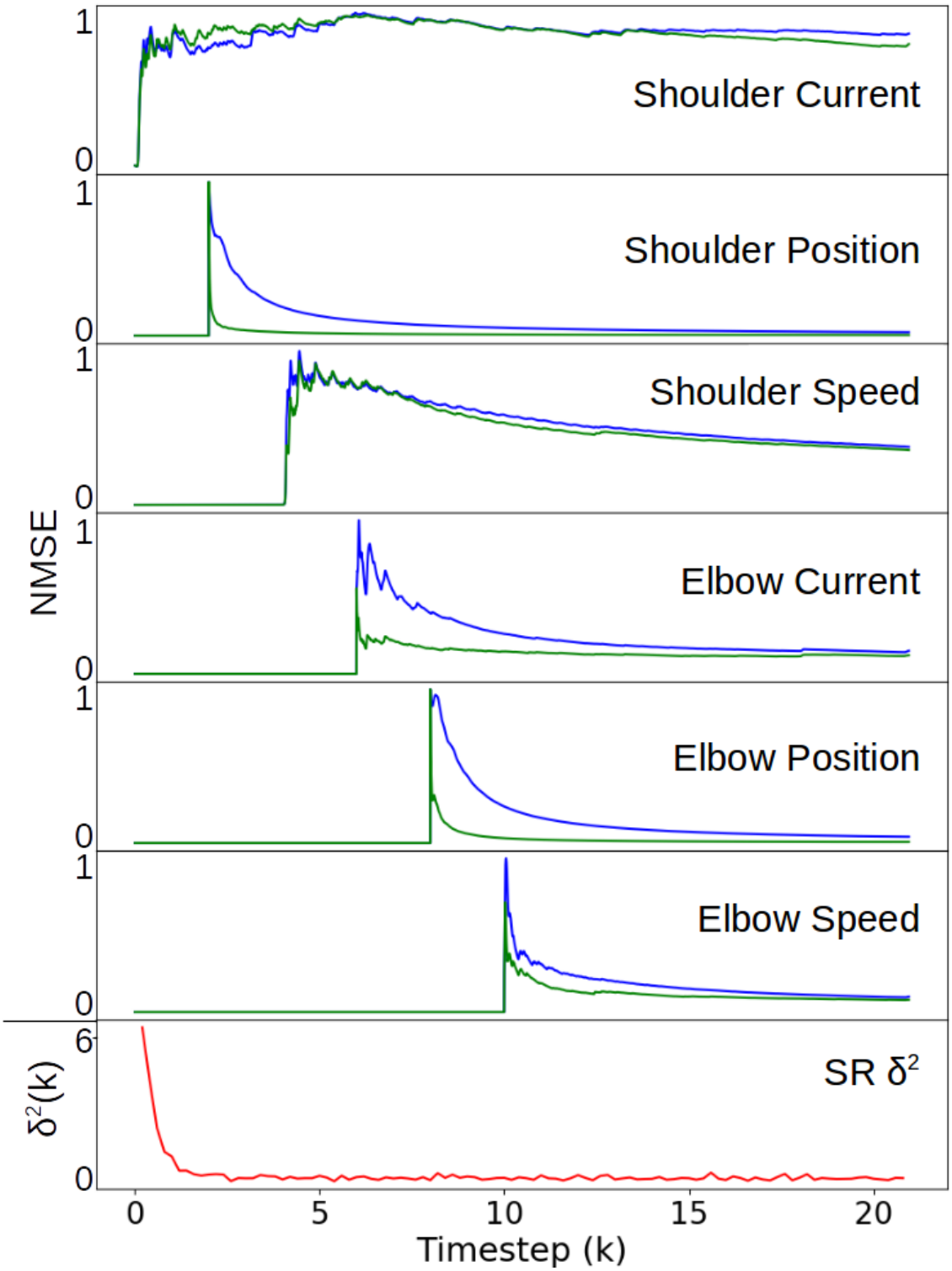


Figure 6.6: The same results as Figure 6.5, but with the NMSE for the individual predictors. Each is normalized from 0 to 1.

to real-world robots.

GVFs were originally proposed as a method for building an agent's overall knowledge in a modular and hierarchical way, representing state as a collection of predictions (Sutton, Modayil, et al., 2011). To date they have been used with hand-coded fixed policies (Modayil, White, et al., 2014; Sherstan, Modayil, et al., 2015) and as state for learned policies (Pilarski, Dick, et al., 2013). GVFs have also been used to track internal statistics such as the variance of the return (Sherstan, Ashley, et al., 2018) and state visitation (Sherstan, Machado, White, et al., 2016). The UNREAL agent (Jaderberg et al., 2017) is a powerful demonstration of the usefulness of multiple predictions; predefined auxiliary tasks, which can be viewed as GVFs, are shown to accelerate and improve the robustness of learning. Work on the construction of GVF models is in its infancy, with the primary approach explored being random generation (Schlegel, White, Patterson, et al., 2018). However, there are related approaches using other predictive frameworks including TD-networks (Makino and Takagi, 2008) and PSRs (McCracken and Bowling, 2006).

There are several ways in which our own results might be improved and extended. Alternative algorithms have been suggested for learning with the SR (Gehring, 2015; Pitis, 2018). In particular, Pitis (Pitis, 2018) shows potential alternatives to: 1) accelerate the learning of value estimates by using the SR and 2) make the one-step signal predictors more robust to noise. These algorithms may further improve the performance of the SR-based prediction approach. Finally, a limitation of the SR is that it is tied to a specific policy. This limitation was relaxed by combining the concept of Universal Value Functions (UVFAs) (Schaul, Horgan, et al., 2015) with the SR in the Universal Successor Representation (USR) (Ma et al., 2018). The USR enables SR-based predictions to generalize across goals (and their corresponding policies). Combining our approach with the USR could further improve learning efficiency and generality.

6.8 Conclusions

In this chapter we showed how the successor representation (SR), although originally introduced for another purpose, can be used to accelerate learning in a continual learning setting in which a robot incrementally constructs models of its world as a collection of predictions known as general value functions (GVFs). The SR enables a given prediction to be mod-

ularized into two components, one representing the dynamics of the environment (the SR) and the other representing the target signal (one-step signal prediction). This allows a robot to reuse its existing knowledge when adding a new prediction target, speeding up learning of the new predictor. We demonstrated this behaviour in both a tabular grid world and on a robot arm. These results suggest an effective method for improving the learning rate and sample efficiency for robots learning in the real world. There are several clear opportunities for further research on this topic. The first is to provide greater understanding into why, for a given fixed step-size, some (few) signals are better predicted directly rather than through the SR. Further, the work in using the SR with function approximation is preliminary and more insight can yet be gained in this setting. Another opportunity for research is to explore using SR-based predictions with state-dependent discount functions. Finally, we suggest that SR-based predictions with deep feature learning (Kulkarni et al., 2016; Machado, Rosenbaum, et al., 2018) and an incrementally constructed architecture would be a very powerful tool to support continual or developmental learning in robotic domains with widespread real-world applications.

Chapter 7

Γ-nets: Generalizing Value Estimation over Timescale

Temporal abstraction is a key requirement for agents making decisions over long time horizons—a fundamental challenge in reinforcement learning. There are numerous reasons why making value estimates at multiple timescales might be useful; recent work has shown that value estimates at different timescales can be the basis for creating more advanced discounting functions and for driving representation learning. Further, predictions at many different timescales serve to broaden an agent's model of its environment. In this chapter we present Γ-nets, a method for generalizing value function estimation over timescale, allowing a given GVF to make predictions for any fixed timescale within its training regime, greatly increasing the predictive ability and scalability of a GVF-based model. The key to our approach is to use timescale as one of the value estimator's inputs. This allows us to compute the prediction target for any timescale. Thus, at every timestep our method trains the network on multiple timescales. We first empirically evaluate Γ-nets on a simple square-wave. Next, we predict sensorimotor signals on a robot arm with linear function approximation. Finally, we empirically evaluate Γ-nets in the deep reinforcement learning setting using policy evaluation on a set of Atari video games. Our results show that Γ-nets can be effective for predicting arbitrary fixed timescales, with only a small cost in accuracy as compared to learning estimators for single timescales. Γ-nets make predictions at many timescales without requiring a priori knowledge of the task, making it a valuable contribution to ongoing work on model-based planning, representation learning, and lifelong learning algorithms.

7.1 Value Functions and Timescale

Reinforcement learning (RL) studies algorithms in which an agent learns to maximize the amount of reward it receives over its lifetime. A key method in RL is the estimation of *value*—the expected cumulative sum of discounted future rewards (called the *return*). In loose terms this tells an agent how good it is to be in a particular state. The agent can then use value estimates to learn a *policy*—a way of behaving—which maximizes the amount of reward received.

Sutton, Modayil, et al. (2011) broadened the use of value estimation by introducing general value functions (GVFs), in which value estimates are made of other sensorimotor signals, not just reward. GVFs can be thought of as representing an agent's model of itself and its environment as a collection of questions about future sensorimotor returns; a predictive representation of state (Dayan, 1993). A GVF is defined by three elements: 1) the policy, 2) the *cumulant* (the sensorimotor signal to be predicted), and 3) the prediction timescale, γ. Considering a simple mobile robot, examples of GVF questions include "How much current will my motors consume over the next 3 seconds if I spin clockwise?" or "How long until my bump sensor goes high if I drive forward?"

Modelling the world at many timescales is seen as a key problem in artificial intelligence (Sutton, 1995; Sutton, Precup, et al., 1999). Further, there is evidence that humans and other animals make estimates of reward and other signals at numerous timescales (Tanaka et al., 2016). This chapter focuses on generalizing value estimation over timescale. Our work can be seen as directly connected to the concept of *nexting*, in which animals and people make large numbers of predictions of sensory input at many, short-term, timescales (Gilbert, 2006). Modayil, White, et al. (2014) demonstrated the concept of nexting using GVFs on a mobile robot. Until now, value estimation has generally been limited to a single fixed timescale. That is, for each desired timescale, a discrete and unique predictor was learned. However, there are situations where we may desire to have value estimates of the same cumulant over many different timescales. For example, consider an agent driving a car. Such an agent may make numerous predictions about the likelihood of colliding with various objects in its vicinity. The agent needs to consider the risk of collisions in both the near term and far term and the relevance of each may change with the speed of the car. If the engineer knew which timescales would be needed ahead of time they could design them

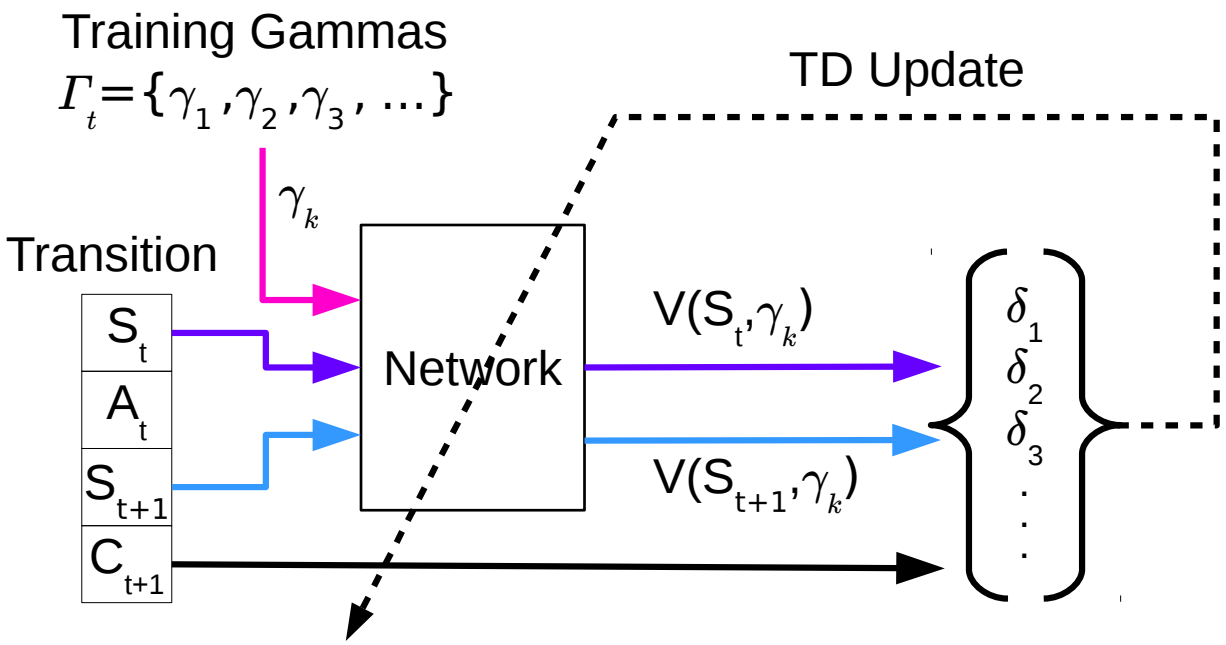


Figure 7.1: **Training γ-nets**. Values are estimated by providing state and timescale, γ, as inputs to the network parameterized by weights $\mathbf{w}$. An agent in state S takes action A and transitions to state S' receiving the new target signal C. On each timestep, t, the agent produces a set of timescales Γ_t on which to train and for each $\gamma_k \in \Gamma_t$ computes values $V(S, \gamma_k; \mathbf{w})$ and $V(S', \gamma_k; \mathbf{w})$. For each γ_k, the TD-error is calculated according to $\delta_k = C + \gamma_k V(S', \gamma_k; \mathbf{w}) - V(S, \gamma_k; \mathbf{w})$. The TD-errors are then collected and used to update $\mathbf{w}$ using a chosen TD learning algorithm, such as TD(λ) or GTD.

into the system, but this is not the case for complex settings.

Here we present a novel class of algorithms which enables the explicit learning and inference of value estimates for any valid fixed discount. The key insights to our approach are: 1) the timescale can be treated as an input parameter for inference and learning and 2) the estimated bootstrapped prediction target for any fixed timescale is available at every timestep. We demonstrate Γ-nets in three policy evaluation settings: 1) predicting a square wave, 2) predicting sensorimotor signals on a robot arm, 3) predicting reward in Atari video games. We also show that Γ-nets can improve policy learning when used as auxiliary tasks in the Atari setting.

The ideas behind our approach are based on work by Schaul, Horgan, et al. (2015) which generalized value estimation across goals by providing a goal embedding vector as input to the value network. In contrast, our approach provides the discount, γ as input. A similar

approach was taken by Xu et al. (2018). Their work presents a meta-learning RL algorithm which learns to automatically adapt the meta-parameters of the algorithm, including the discount. The goal of their algorithm was to automatically determine the best return to use for learning. They found that providing these meta-parameters as input to both their value and policy networks significantly improved performance in learning policies on many Atari games.

7.2 Background

We model the environment as a Markov Decision Process. At each timestep t the agent, in state $S_t \in \mathcal{S}$, takes action $A_t \in \mathcal{A}$ according to policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ and transitions to state $S_{t+1} \in \mathcal{S}$ according to the transition probability $p(\cdot|S_t, A_t)$. In the traditional RL setting the agent receives a reward $R_{t+1} \equiv R(S_t, A_t, S_{t+1}) \in \mathbb{R}$. The agent tries to learn a policy which maximizes the cumulative reward it receives in the future, which is defined as the return: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$. In the case of GVFs we simply substitute our signal of interest, the cumulant, C for reward, R. The term $\gamma \in [0, 1)$ is referred to by several names including the timescale, the continuation function and the discount; it represents the amount of emphasis applied to future rewards and is the focus of this chapter.

A value estimate is simply the expectation of the return: $V_\pi(s) = \mathbb{E}_\pi\big[G_t|S_t = s\big]$. Temporal-difference (TD) learning is a common class of algorithms used in RL for learning an approximation of value (Sutton and Barto, 2018). Estimation weights are typically trained by semi-gradient descent using the TD-error: $\delta_t = C_{t+1} + \gamma V_\pi(S_{t+1}) - V_\pi(S_t)$.

While simple domains can be represented using tabular lookup, complex settings in which the state space is very large or infinite must use function approximation (FA) methods to estimate the value as $V(s; \mathbf{w})$, where $\mathbf{w}$ is a set of weights parameterizing the network. Function approximation has the advantage that states are not treated independently, but rather, a learning step updates related states as well, allowing for generalization across state-space.

7.3 Increased Representational Power

One of the advantages of TD algorithms is that they allow the agent to bootstrap estimation of the return from its existing estimates. However, this limits a single predictor to capturing

only returns with geometric discounting. By combining predictions at several timescales it is possible to capture returns of different shapes such as the example shown in Figure 7.2. In fact, Fedus et al. (2019) used geometrically discounted value estimates as a basis to estimate hyperbolically discounted returns. Figure 7.3 shows the different returns of a single reward of magnitude 1.0 occurring at different distances in the future. We see that shorter timescales are unaware of distant rewards, as expected.

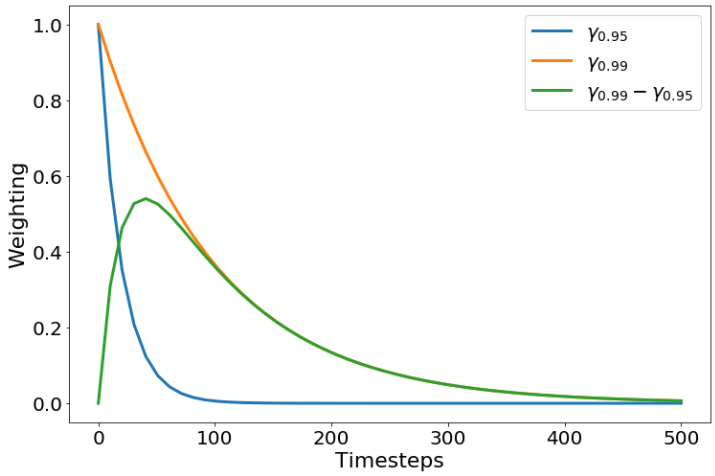


Figure 7.2: Capturing non-geometric returns (green) by taking the difference of two predictions at different timescales.

7.4 Generalizing over Timescale

Our goal is to be able to predict the value function for any discount γ. While the GVF specification allows for γ that are a function of the transition, here we focus solely on the case of fixed timescale. To achieve that goal, we propose Γ-nets: an architecture for value functions that operates not only on the state, but also the desired target discount factor γ_k (see Figure 7.1). The keys to generalizing over timescale with Γ-nets are: 1) make value a function of γ by providing γ as one of the network inputs, i.e., $V(s, \gamma_k)$, and 2) train on many timescales simultaneously.

On each training step, t, the agent produces a new set of training timescales Γ_t. In our experiments this set is produced by sampling over a distribution of timescales, but other

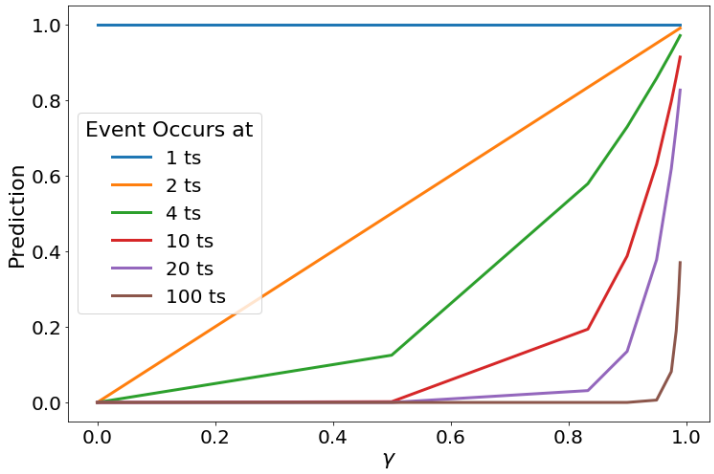


Figure 7.3: For each series a single event, value=1.0, occurs at some time in the future. Different timescales show different responses to these events. This suggests that different prediction timescales can enhance an agent's state representation.

approaches are possible. For each $\gamma_k \in \Gamma_t$ the TD-error is:

$$\delta_{t;\gamma_k} = C_{t+1} + \gamma_k V_\pi(S_{t+1}, \gamma_k) - V_\pi(S_t, \gamma_k). \tag{7.1}$$

Gradients are then computed for all $\gamma_k \in \Gamma_t$ and applied to update the network weights. By training over many different timescales simultaneously, our network learns to generalize over timescale.

Choosing Γ_t must be done with care. A naive approach might uniformly sample $\gamma_k \in [0, 1)$. However, value functions change non-linearly with γ. To illustrate this property, consider that γ can be viewed as the probability of continuation, allowing us to derive the expected number of timesteps until termination of the return as (see Sherstan (2015) for a derivation):

$$\tau = \frac{1}{1-\gamma}. \tag{7.2}$$

Table 7.1 shows selected values of γ and their corresponding values of τ. The relationship between γ and τ is non-linear for large values of γ (Figure 7.4). Thus, naively drawing γ_k from a uniform distribution would tend to favor very short timescales. Conversely, drawing uniformly from τ would put little emphasis on short timescales. While the best method for selecting γ_k for training is outside the scope of this chapter, we provide some comparisons

in our experiments. Note that throughout this chapter we will refer broadly to the word *timescale* for which we will use the parameters γ or τ as appropriate. It should be assumed that these terms can be used interchangeably using Eq. (7.2).

Table 7.1: Expected Timesteps

$\boldsymbol{\gamma}$	$\boldsymbol{\tau}$
0	1
0.5	2
0.8	5
0.9	10
0.95	20
0.975	40
0.983	60
0.9875	80
0.99	100

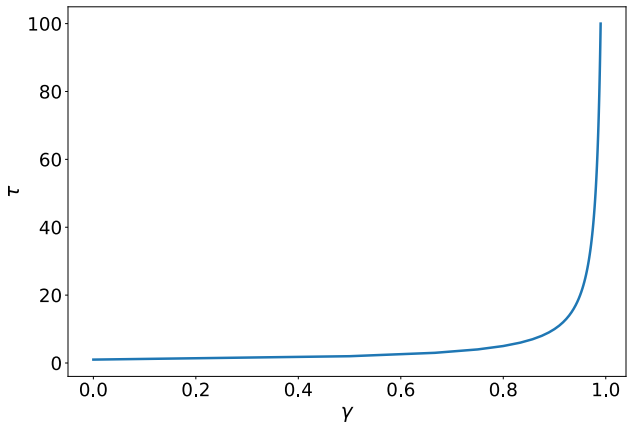


Figure 7.4: Non-linear relationship between discount γ and prediction length in expected timesteps.

The representation of timescale used for input to the network may affect the network's ability to represent different timescales. The γ scale compresses long timescales but spreads short ones and in the τ scale we have the opposite effect. Thus, providing both γ and τ as input may allow for good discrimination at all timescales.

Finally, the magnitude of returns at different timescales can be very different. Larger

returns can produce larger errors and corresponding larger gradients, which can effectively dominate the network weights. In general it is the longer timescales which will produce larger magnitude returns, but returns can be constructed for which the opposite is true. To prevent large magnitude returns from dominating the network weights we need to scale the returns in some way. We want to look for a general solution as we may not know beforehand which timescales are most important and thus seek a way to balance accuracy for all timescales. A general approach is given by van Hasselt et al. (2016), in which they continually normalize the target to have a mean of 0 and variance of 1. This allows them to handle rewards of varying magnitude. Here, we take a simpler approach focusing on keeping the magnitude of the returns, as a function of timescale, in the same ballpark, by learning the value of a scaled cumulant: $f(s, \gamma_k; \mathbf{w}) = \mathbb{E}_\pi[\sum_{t=0} \gamma_k^t (1-\gamma_k) C_{t+1} | S_0 = s]$. This will scale the loss by timescale and should result in smaller network weights. However, the resulting prediction must then be rescaled by dividing by $(1-\gamma_k)$. However, if we instead redefine our value estimator as

$$V(s, \gamma_k; \mathbf{w}) = \frac{f(s, \gamma_k; \mathbf{w})}{(1-\gamma_k)} \tag{7.3}$$

then we can simply scale the TD loss by $(1-\gamma_k)$. In the following we show this derivation for n-step returns. The TD-error for the n-step scaled cumulant is:

$$\delta_{t;\gamma_k} = (1-\gamma_k)(C_{t+1} + \gamma_k C_{t+2} + \ldots + \gamma_k^{n-1} C_{t+n}) + \gamma_k^n f(S_{t+n}, \gamma_k) - f(S_t, \gamma_k).$$

But, if we substitute in V from Eq. 7.3 we have:

$$\begin{aligned}
&= (1-\gamma_k)(C_{t+1} + \gamma_k C_{t+2} + \ldots + \gamma_k^{n-1} C_{t+n}) + \gamma_k^n (1-\gamma_k) V(S_{t+n}, \gamma_k) - (1-\gamma_k) V(S_t, \gamma_k) \\
&= (1-\gamma_k)(C_{t+1} + \gamma_k C_{t+2} + \ldots + \gamma_k^{n-1} C_{t+n} + \gamma_k^n V(S_{t+n}, \gamma_k) - V(S_t, \gamma_k)).
\end{aligned}$$

This results in scaled losses and their scaled gradients. This scaling can then be applied at either the loss or gradient level.

7.5 Experiments

We first provide two proof of concept demonstrations using linear function approximation. The first on a square-wave signal, which is easily understood. The second on a robot arm.

Next we empirically evaluate Γ-nets in a deep learning setting by looking at performance on Atari games.

7.5.1 Square-wave

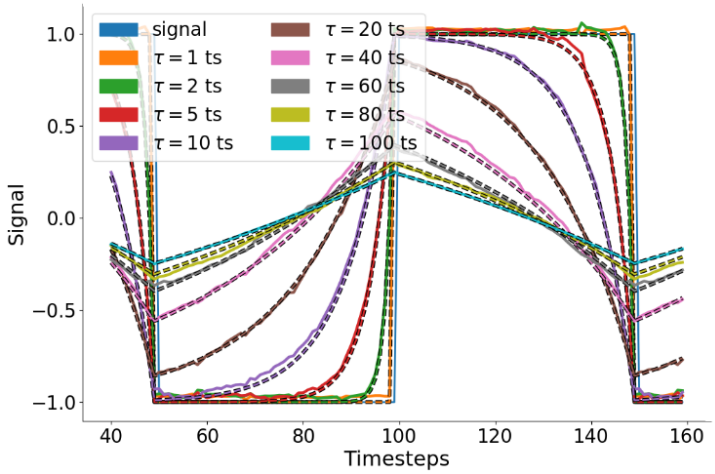


Figure 7.5: **Square-wave Predictions.** Predictions (solid) against the true return (dashed) after 50k ts of training using both γ and τ as input, scaling the loss and drawing two γ_k each from γ and τ scales plus $\tau = 1, 100$. For display purposes all predictions are normalized by $(1 - \gamma)$. We see good accuracy across all timescales.

Our target signal was a repeating square wave 100 timesteps in length with a magnitude of $\{-1, 1\}$ (Figure 7.5). Inputs were normalized and then tilecoded (Sutton and Barto, 2018) with 20 tilings of width 1.0, 20 tilings of width 0.5 and 30 tilings of width 0.1. Tiling positions were randomly shifted by small amounts at the time of initialization for each run. Value estimates were computed using linear function approximation on the output of the tilecoding and the final layer of weights were updated using TD(0) (Sutton and Barto, 2018) (The algorithm used for this experiment is given in Algorithm 2). We also evaluated the impact of loss scaling. Unless otherwise stated: 1) timescale inputs were given on both the γ and τ scales simultaneously, 2) Γ_t was 6 elements long, with $\tau \in \{1, 100\}$ always included and two additional timescales drawn uniformly from each of the γ and τ timescales, 3) loss scaling was used. Results are shown in Figures 7.6-7.8. Each training run lasted for 50k timesteps and for each series 100 different runs were made. We show the normalized errors as a function of the prediction timescale, given on the τ-scale. Results are averaged over the

last 5k timesteps. For each τ we normalize by the maximum mean error across the series in the plot. Error bars indicate standard error.

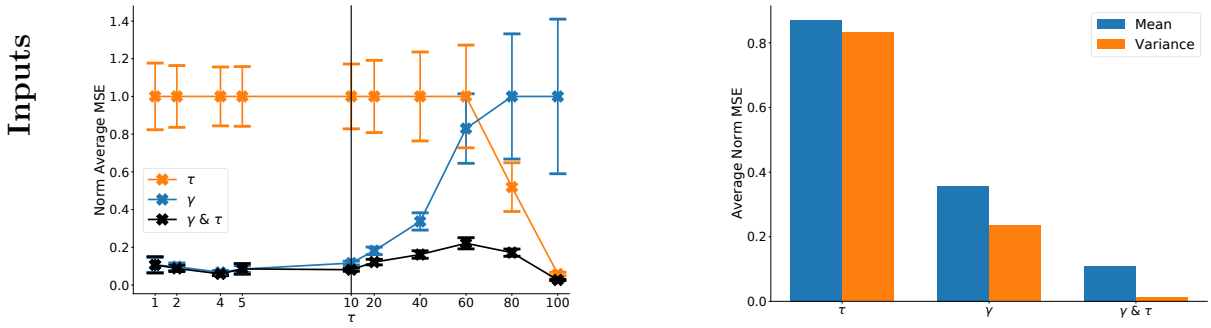


Figure 7.6: **Inputs (Square Wave).** Comparing the effect of using different timescale representations as input. As expected, providing timescale as γ did better than τ on the short timescales, but worse on the longer timescales, although this cross over occurred at a much longer timescale than expected. Providing both γ and τ did the best of all, producing the lowest errors across all probe timescales as well as providing the lowest variability. **Error bars for the square-wave plots indicate standard error.**

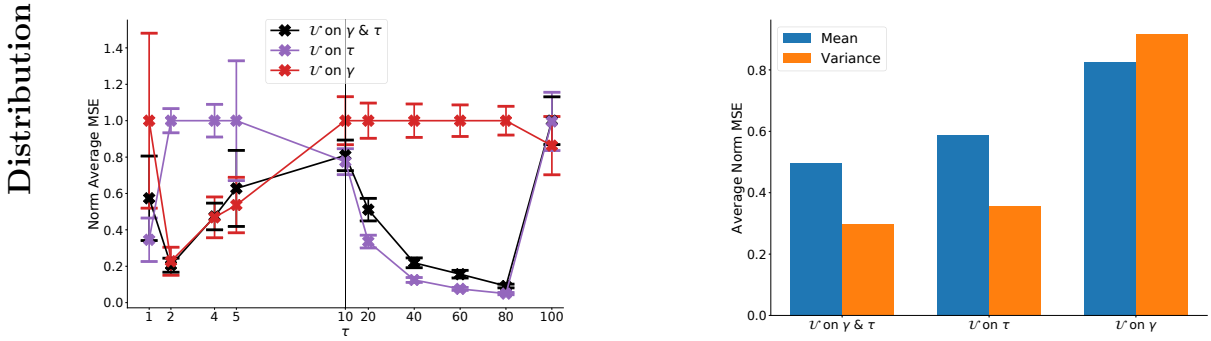
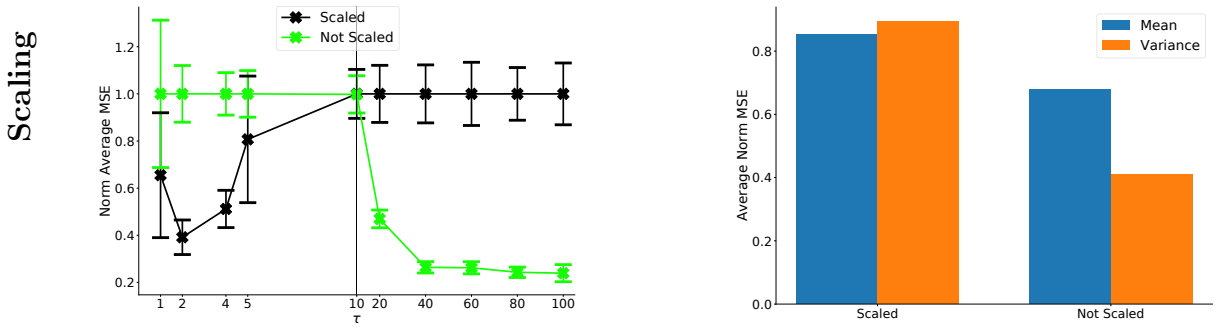


Figure 7.7: **Distribution (Square Wave).** Here we compare the effects of drawing γ_k from different distributions. As previously discussed, Γ_t was 6 elements long, always including $\tau \in \{1, 100\}$, and sampling the additional 4 γ_k. Excluding $\tau \in \{1, 100\}$ we see that drawing all γ_k from the γ scale performs better than drawing all from τ scale at shorter timescales, but does worse at longer timescales. Drawing half from each tends to follow the lower errors at all the timescales.



(a) MSE (b) Avg Norm MSE

Figure 7.8: **Scaling (Square Wave)**. We compare the effects of scaling the cumulant. Here we see that scaling does improve performance on the shorter timescales, but causes worse performance on the longer ones.

Algorithm 2 Generalization over γ with TD(0)

Input: Feature representation $\phi \in \mathbb{R}^n$, policy π, and step-size α
Output: Vector $\mathbf{w}$.
Initialize $\mathbf{w} \in \mathbb{R}^n$ arbitrarily
while S' is not terminal **do**
  Observe state S, take action A selected according to $\pi(S)$, and observe a next state S' and cumulant C
  Pick a set of γ_k to train on:
  $\Gamma \leftarrow \gamma SelectionFunction(Terminal = False)$
  $\boldsymbol{\Delta} \leftarrow \mathbf{0}$; Zeros vector, length n
  for γ_k in Γ **do**
    $\delta = C + \gamma_k \phi(S_{t+1}, \gamma_k)^\top \mathbf{w} - \phi(S_t, \gamma_k)^\top \mathbf{w}$
    $\boldsymbol{\Delta}$ += $\delta \phi(S_t, \gamma_k)$
  end for
  $\mathbf{w} = \mathbf{w} + \alpha \boldsymbol{\Delta}$
end while
$\Gamma \leftarrow \gamma SelectionFunction(Terminal = True)$
for γ_k in Γ **do**
  $\delta = C - \phi(S_t, \gamma_k)^\top \mathbf{w}$
  $\boldsymbol{\Delta}$ += $\delta \phi(S_t, \gamma_k)$
end for
$\mathbf{w} = \mathbf{w} + \alpha \boldsymbol{\Delta}$

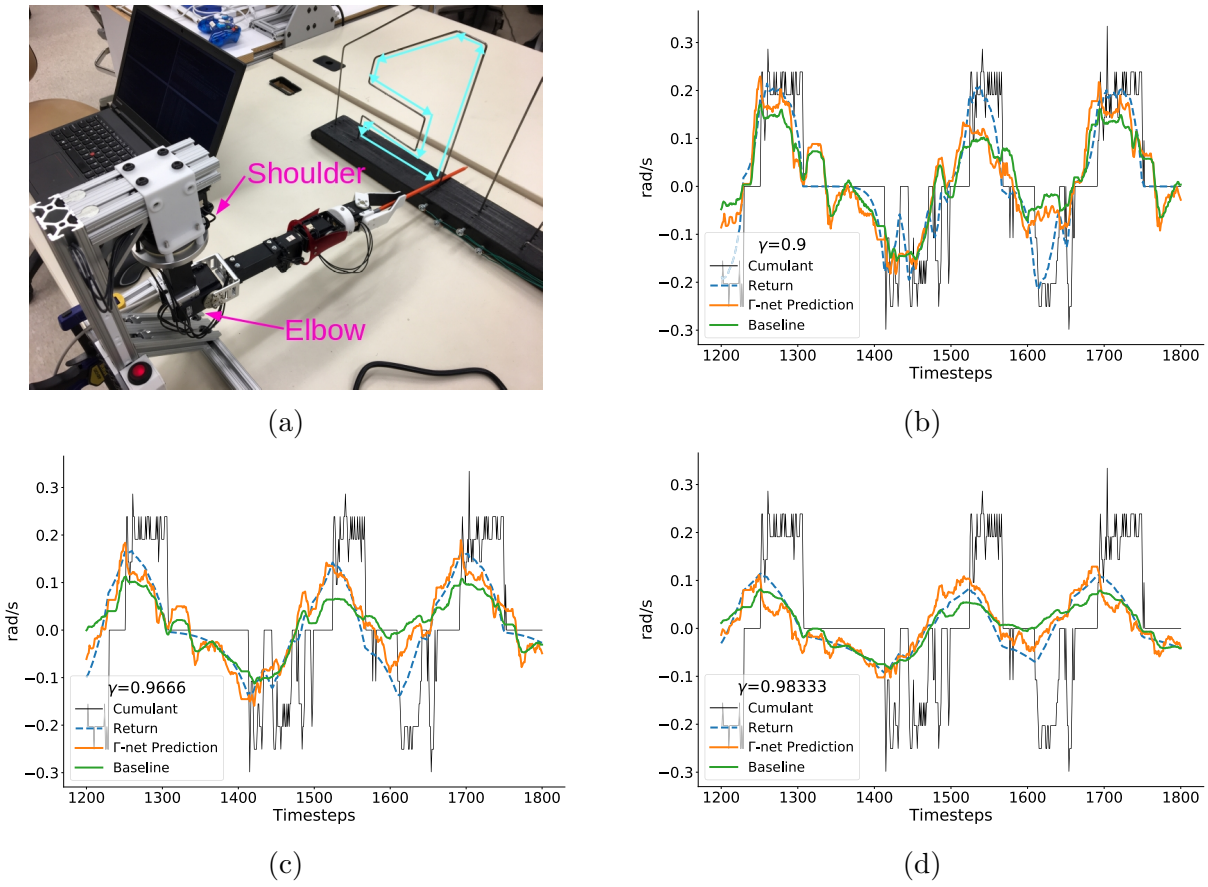


Figure 7.9: **Robot Arm. a**) A user controls the shoulder and elbow joints of a robot arm via joystick to move a rod counter-clockwise around a wire maze. **b-d**) Predictions of the speed of the shoulder joint. Predictions and returns are scaled by $(1-\gamma)$ for display purposes. Timescales correspond to ∼0.33, 1.0, 2.0 s. Γ-net predictions (orange) perform similarly to the baseline (green) in matching the return (blue).

7.5.2 Predictions on a Robot Arm

In this experiment a human operated the shoulder rotation and elbow flexion joints of a robot arm by joystick (this is the same as described in Chapter 6). The task was to maintain contact between a rod held by the robot and the inside of a wire maze while moving in a counter-clockwise direction (Figure 7.9a). Fifty circuits of the maze were completed in approximately 12 minutes. Network inputs were the normalized positions of the shoulder and elbow servos as well as both γ and normalized τ. Inputs were tilecoded (Sutton and Barto, 2018) with 100 tilings of width 1.0 into a space of 2048 bits and a bias unit was added giving a feature vector of 2049 bits. Value estimates were computed by linear function approximation (LFA) and trained by TD(0). On each timestep Γ_t was generated from $\tau \in [1, 100]$ ts. The upper

and lower bounds were included in the set and one γ and 29 τ were sampled uniformly from their respective scales for a total of 32 timescales. More emphasis was placed on sampling from the τ scale because of the relatively high update rate (30 Hz, 1 ts $\sim$ 0.03 ms). Thus, the likely important timescales will be above $\gamma = 0.9$. Loss prescaling was used. We used a step-size of 0.1 divided by the number of active features. The step-size was linearly decayed to zero over the course of the training set. A baseline predictor with a fixed timescale was also trained using the same parameters as the Γ-net excepting the inclusion of timescale input.

Figure 7.9 shows the Γ-net predicting shoulder joint speed at several timescales. Table 7.2 shows the cumulative sum of absolute error between the predictor and the return over the whole dataset after training. With this configuration the Γ-net outperformed the baseline for most of the timescales tested. For $\gamma = 0.99$ the baseline performed slightly better.

Table 7.2: **Robot Arm.** Cumulative absolute error over the entire dataset after training. Lower is better.

γ	Γ-net	Baseline
0.9	**1025**	1124
0.9666	**602**	822
0.98333	**379**	440
0.99	273	**253**

7.5.3 Atari Environment

We examined the performance of Γ-nets under policy evaluation in the Atari Arcade Learning Environment (ALE) (Bellemare, Naddaf, et al., 2015). The agent's policy was trained using the Dopamine project's (Castro et al., 2018) implementation of the Rainbow agent (Hessel et al., 2018) (also, see Section 2.3.3), which uses the same network architecture as the DQN agent (Mnih, Kavukcuoglu, et al., 2015), but adds prioritized replay (Schaul, Quan, et al., 2016), n-step returns, and distributional representation of the value estimates (Bellemare, Dabney, and Munos, 2017). The architecture used in shown in Figure 7.10.

The primary results presented are for the game Centipede with a Rainbow agent trained for 25 M frames, which we will refer to as *Centipede@25M*. Additional Atari games were evaluated using agents trained for 200 M frames, which we will refer to as *Atari@200M*. These agents were included as part of the Dopamine package and trained according to the

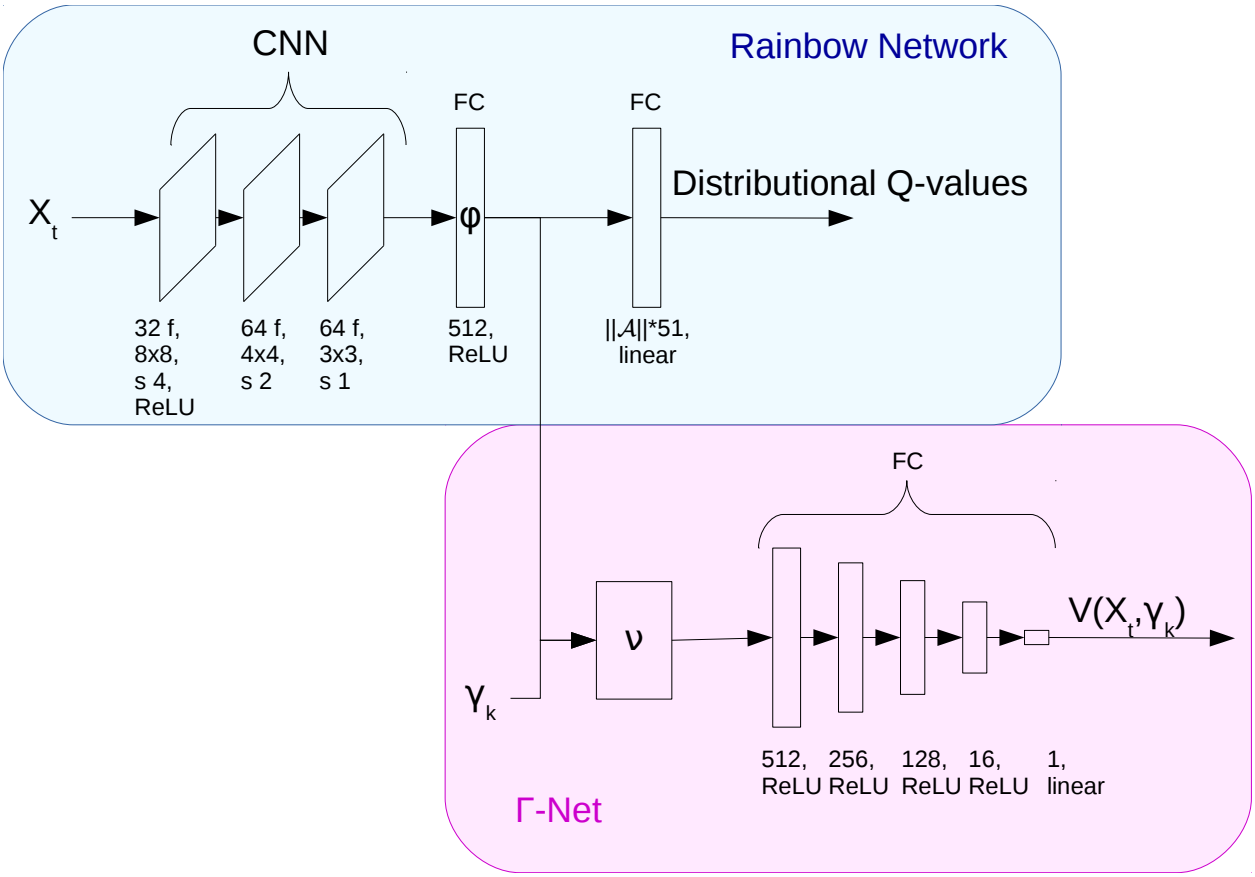


Figure 7.10: Architecture used for the policy evaluation experiments. The Rainbow Network, from Dopamine, is used to generate episodes of data where we save the feature vector, ϕ, to file. To train the Γ-net we then read in these files, store them in a prioritized replay buffer and sample from this replay buffer. The feature vector, ϕ, is then combined with the γ_k using the embedding function ν which acts as input to the Γ-net.

specifications given in Castro et al. (2018).

Figure 7.11 shows predictions on the early transitions of a single episode. For this episode the expected return was estimated by running 2000 Monte Carlo rollouts from each state visited along the way (dashed lines). The solid lines indicate the Γ-net predictions after training for 20M frames (using the *direct* configuration which is described in following sections).

Training

The prediction networks were trained using samples of transitions generated by pretrained policies. Agents select actions using ϵ-greedy over their Q-values. During policy training $\epsilon = 0.001$, but for generating the samples used for training the Γ-nets we use an evaluation

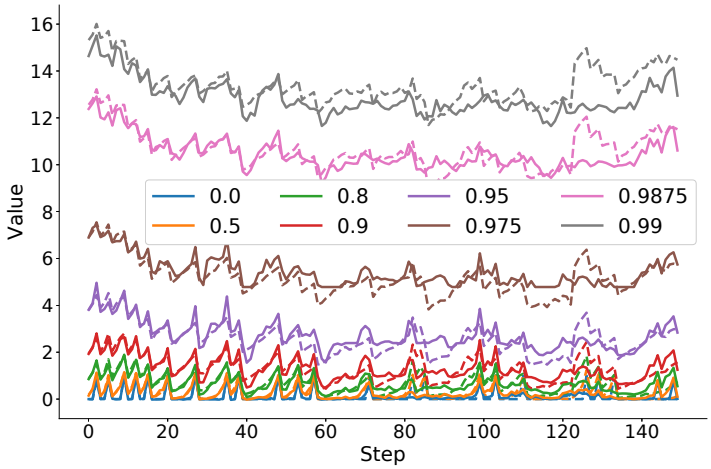


Figure 7.11: Predictions on Centipede@25M for different γ from the start of a single episode. Γ-net predictions are shown in solid lines and the expected return, produced by Monte Carlo rollout, is shown by the dashed lines.

mode where $\epsilon = 0.0001$. Transitions were generated sequentially and the environment was reset at the end of each episode or 27,000 steps, whichever came first. These transitions were saved to file in sequence and for each experiment they were reloaded in the same order. For each transition, we saved the reward as well as the activation of the final core layer of the agent's network ϕ, which serves as the input to the Γ-nets. The Γ-net network was composed of five fully-connected layers of sizes [512, 256, 128, 16, 1], with all but the final layer using ReLU activation. Training of the Γ-nets proceeded as if the data was generated in an online fashion, as would be the case during policy learning. That is the agent would read in transition samples from the file, add them to a prioritized replay buffer, and then train by sampling from the replay buffer. When a new sample was added to the buffer it was given the highest level of priority so that its probability of being sampled was high. Like the policy training we train on a batch of sampled transitions, using n-step returns. To update the priorities for a given sample in the batch we use the maximum squared loss across Γ_t.

A Γ_t of size 8 was used, which always included lower and upper bounds of $\tau = \{1, 100\}$. An additional 6 γ_k were drawn on each timestep. Unless otherwise stated the sampling was done by drawing 3 γ_k uniformly from the γ scale on $[0, 0.99)$ and the τ scale on $[1, 100)$ (for τ we drew from the integer scales, rather than float). Each network was trained for 20 M frames with network weights saved every 500k frames.

Parameter	Value
Input dim	84x84 pixels
ϕ dim	512 nodes
Replay buffer size	100k transitions
Batch size	32 samples
n-step	4 transitions
Min-replay history	20k transitions
Frameskip	4 frames/transition
Training rate	1/4 transitions
Sync interval	10k training steps
Sticky-actions	0.25
Terminal on life loss	False
Max steps per episode	27k transitions
Consecutive frame pooling	True
ϵ-greedy: policy learning	0.001
ϵ-greedy: evaluation	0.0001
Adam optimizer: Step-size (learning rate)	$6.25e^{-5}$
Adam optimizer: eps	$1e^{-8}$

Table 7.3: Parameters

Various parameters are indicated in Table 7.3.

A brief sweep was made over the step-size parameter (also referred to as *learning rate*) for the Centipede@25M policy. Sweeps were made over the probe timescales as well as over various variants of the Γ-net for 3 seeds each. The values tried were: $6.25e^{-4}, 6.25e^{-5}, 6.25e^{-6}$. It was found that, almost universally, the value $6.25e^{-5}$ gave the lowest error when errors were aggregated over all probe timescales. This is also the step-size used in training the Rainbow agent. This step-size value was used for all reported experiments. Note that these sweeps were done on Centipede@25M experiments only.

Dopamine's implementation of the prioritized replay buffer used fixed discounting for a single timescale. Thus, we needed to modify this implementation to return the n-step transitions and then apply discounting afterwards.

We use a frame skip of 4, meaning that when an action is sent to the environment it is executed 4 times in a row and the resulting final frame is returned as observation. The implementation also uses frame pooling in which a max pooling is taken over the last 2 consecutive frames in order to deal with flicker in the rendering of the game images. We used sticky-actions with a probability of 0.25. This means that when an action is sent to the environment there is a 25% chance that the environment will use the previous action instead.

Every reset of the ALE environment restores the environment to the same initial state. State transitions are deterministic. The policy was trained with an ϵ-greedy value of 0.001, but for evaluation transitions were generated with ϵ reduced to 0.0001. Thus, during evaluation the largest source of stochasticity is due to the sticky-actions. Further, the agent sees the early states of the episode more frequently than the later states. Like the policy training, one training update was performed for every 4 steps in the environment. Since every step in the environment corresponds to 4 frames a training update was performed every 16 frames.

To train the sampled batch of transitions on Γ we pair the samples with each γ_k. Thus, for a batch size of 32 sampled transitions and a Γ of 8 the effective batch size is 256. This does add some additional computation time to the process, but this is also affected by the quality of the implementation. When sampling from the τ scale for Γ_t we used the integer scale rather than float.

Like the policy we use a target network which periodically copies weights from the online network; it is the online network which is updated on each training step and the target network which is used for bootstrapping. Note TD learning is typically trained using a semi-gradient approach in which the gradients are not computed with respect to the bootstrapping.

Evaluation

To evaluate predictive accuracy we created a set of evaluation points for each game. These were generated by running the agent in evaluation mode over multiple episodes. At the start of each episode an offset was randomly chosen between $[10, 100)$ steps. Then, starting at the offset, the state of the environment and agent were saved every 30 steps (120 frames with 4 frame frameskip). For Centipede@25M a total of 269 evaluation points were created in this way including the episode start state. From each of these evaluation points we ran 1000 episodes till termination and then computed the average return. These were used as the baseline against which we computed our prediction error. To compute the prediction error for a given evaluation point we restored the agent's and environment's state and recorded the network's predictions for the probe timescales $[1, 2, 5, 10, 20, 40, 60, 80, 100]$ ts. For comparison, we trained fixed timescale networks for all the probe timescales (plotted in fuchsia). These networks used exactly the same architecture as the Γ-net, but did not provide timescale as input to the network and only trained on the single fixed timescale. These probe networks also used loss scaling.

We use a reference configuration of the Γ-net across the different plots. We plot this series in black and refer to it as *direct* although the legends may give it a different label to call out the significance of its configuration for a given comparison. For this configuration both γ and τ were provided as inputs to the network. Additionally, Γ_t was populated by drawing samples from both γ and τ scales and loss scaling was used. For each of the other configurations only a single setting was modified from this reference.

Plotting

We focus our evaluation on the steady-state performance of the network, computing averages over the last 5 M frames of the 20 M frame runs (with evaluation at every 500k frames). Mean-squared error (MSE) for each experiment is presented as a function of the evaluation probe timescale given in τ (Ex. Figure 7.12a). For each τ we normalize across the different series by the largest mean error. Thus, the largest mean error for each τ is shown as 1.0. We do this to be able to clearly show results for all the different timescales in a single plot despite the large differences in magnitude. As a result, series can only be directly compared within a plot, not across plots. To rank each for comparison we provide a bar chart (Ex. Figure 7.12b) which averages the normalized means and normalized variances of the MSE. That is, we take the normalized mean MSE for each τ and average across all τ. Likewise we take the variance at each τ, normalize it by the maximum variance for each τ and take the average across all τ. Note that averaging this way is a biased approach in that it is dependent on what probe τ are used. For example, if we took many large τ and few small ones then our results would give more weight to the large τ. In practice, the importance of errors for different timescales will be task dependent.

While conducting parameter sweeps it was observed that a particular network configuration might produce the lowest value of MSE but not actually be predictive. In this case the network would learn to always output a fixed value which captured the mean of the expected returns. Thus, we adopted a two step evaluation process. First, we took the evaluation points and concatenated them in sequence. We then computed the correlation between their expected returns and the predicted returns made by the network. If a configuration had a positive correlation then it would be considered for comparison with other architectures. We have also included the plots of correlation by probe timescale (Ex. Fig 7.12c). Correlation values are easily interpreted with the maximum (best) value of 1. This tells us

how closely the shapes of the target sequence and the prediction sequence match.

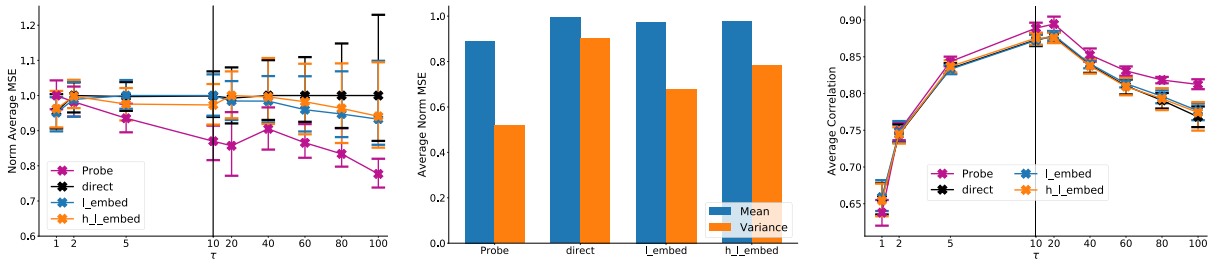
All series are an average over 6 seeds and the shading indicates max and min values. Note that, due to the high degree of overlap in many of the figures, color printing is required to discern individual series. Plots taken with respect to τ are produced by combining two different x-axes, allowing us to make both short and long timescales discernible. This split occurs at $\tau = 10$ and is indicated by the vertical black line.

For the *Atari@200M* results we also included learning curves (rightmost column). These learning curves take an average of normalized MSE taken across all evaluation timescales. For each timescale we normalize each of the series by the largest MSE of any series for that timescale. Then for each series we average the normalized MSE across all the timescales. As before, shaded areas indicate max and min.

While our evaluation method seeks to discern differences in performance due to the various configuration, in reality most configurations perform similarly. In order to rank configurations we first considered the MSE and then variance.

Embedding Comparison.

We compare methods for combining the timescale inputs with the agent's features, ϕ, using an embedding vector $\nu = \nu(\phi, \gamma)$ (Figure 7.12). The ***direct*** embedding performs a concatenation, $\nu = [\phi, \gamma]$. Xu et al. (2018) learned a vector, $\xi(\gamma)$, of size 16 which was concatenated with ϕ, which we refer to as ***l_embed***. We also considered a Hadamard embedding in which a learned vector, $\xi(\gamma)$, the same length as ϕ, was combined using element-wise multiplication with ϕ, that is $\nu = \phi \odot \xi$ (***h_l_embed***). Finally, we considered a matrix multiplication approach in which the timescales were given as inputs to a fully connected layer whose output was a square matrix, $\Xi(\gamma)$, with dimensions the same size as ϕ. The embedding was then formed by matrix multiplication: $\nu = \phi^\top \Xi$. All embeddings were trained as part of the end-to-end training of the network against the prediction loss. We found little difference between the approaches in terms of their MSE or correlation. Overall the linear embedding appears the best choice based on its lower variance, but this did not hold universally for the other games evaluated (Figure 7.15). Learning and computation were both slower with the matrix multiplication approach (Figure 7.13) and linear activations were generally slightly better than ReLU (Figure 7.14).



(a) MSE (b) Avg Norm MSE (c) Correlation

Figure 7.12: **Embedding Comparison.** Several approaches for adding the timescale dependency to the network were investigated. **direct)** Concatenates the timescale with ϕ. **l_embed)** Timescale is input to a fully connected layer of length 16 with linear activation whose output is concatenated with ϕ. **h_l_embed)** Timescale is input to a fully connected layer the same length as ϕ with linear activation and then combined with ϕ by element-wise multiplication. The *l_embed* approach appears to be slightly better due to its lower variance. **Note that, unlike the square-wave plots, error bars in the Atari plots indicate max-min.**

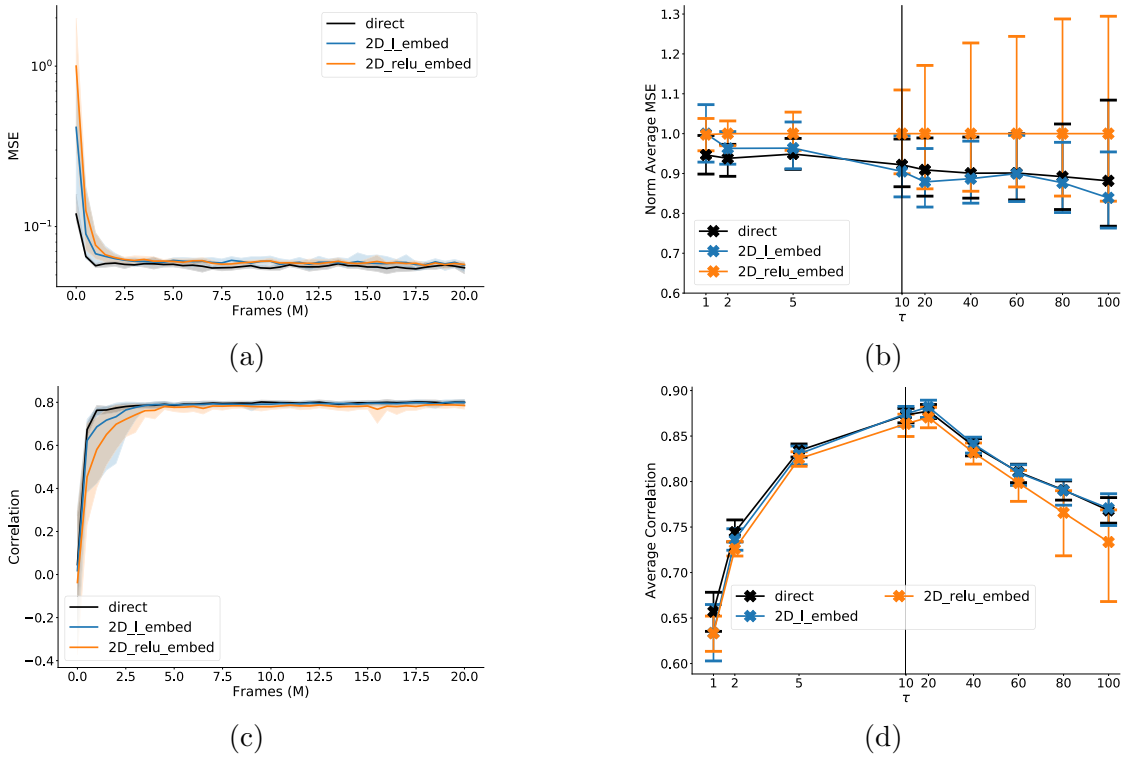


Figure 7.13: **Matrix Embedding.** Timescale embedding was performed by matrix multiplication, that is $\nu_{t;k} = \phi_t^\top \Xi_k(\gamma_k)$. Where the size of Ξ was chosen such that the multiplication produced an output vector the same size as ϕ. That is, $||\phi|| = 512$ and $||\Xi|| = 512 \times 512$. The series *2D_l_embed* and *2D_relu_embed* use the matrix multiplication approach with either a linear or ReLU activation function. The matrix multiplication approach tends to learn much slower than the direct, with markedly reduced performance at the higher timescales. Further, consistent with the other embedding approach, ReLU activation performs worse than a linear one. Additionally, training the 2D models was computationally expensive and training was noticeably longer.

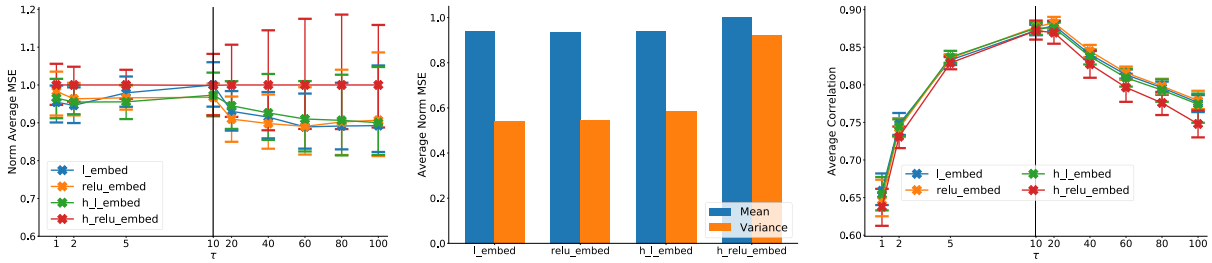
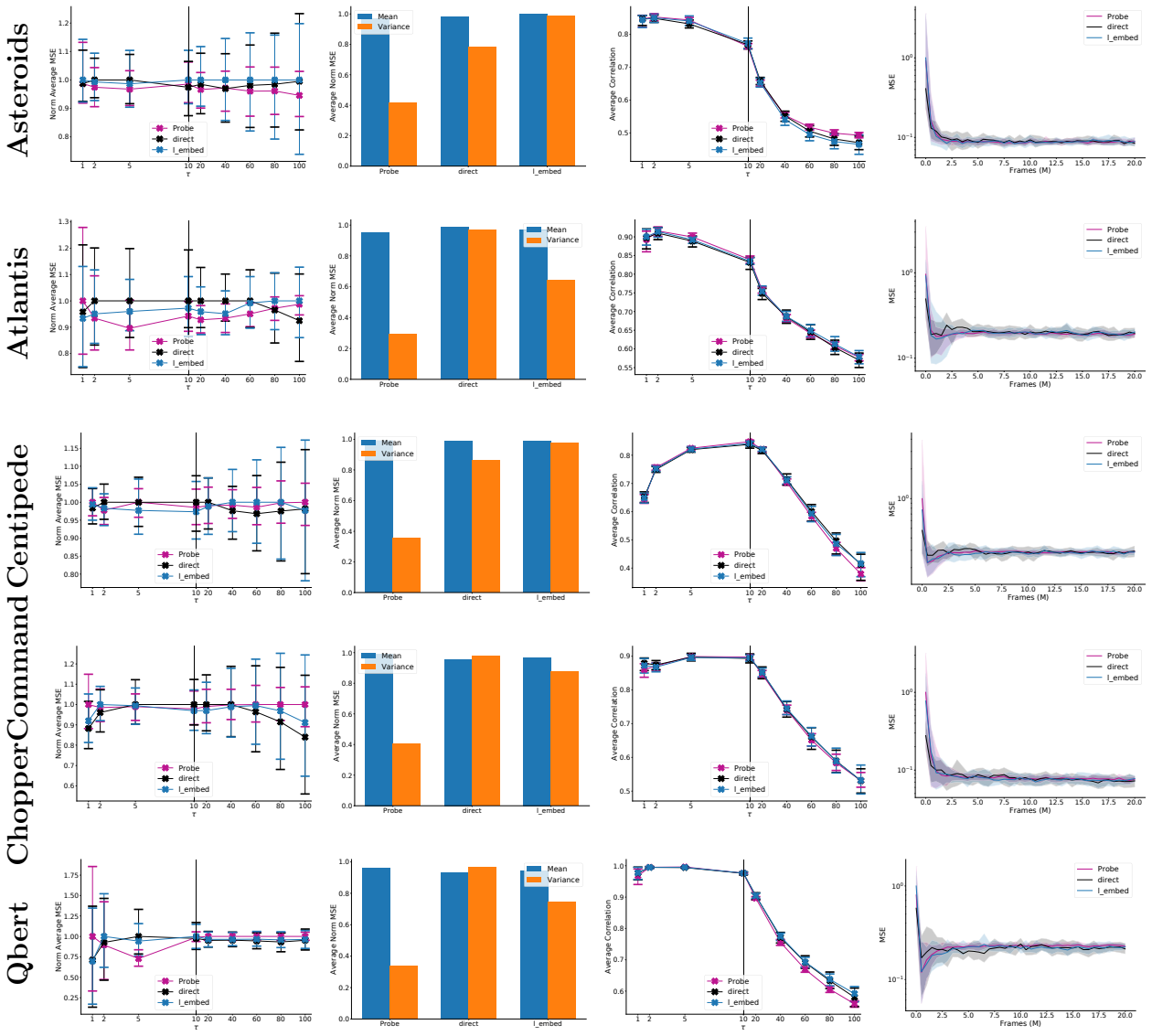


Figure 7.14: **Linear and ReLU Embedding Activations.** If we include Figure 7.13 it generally appears that linear activations are better than ReLU for the embedding layers.

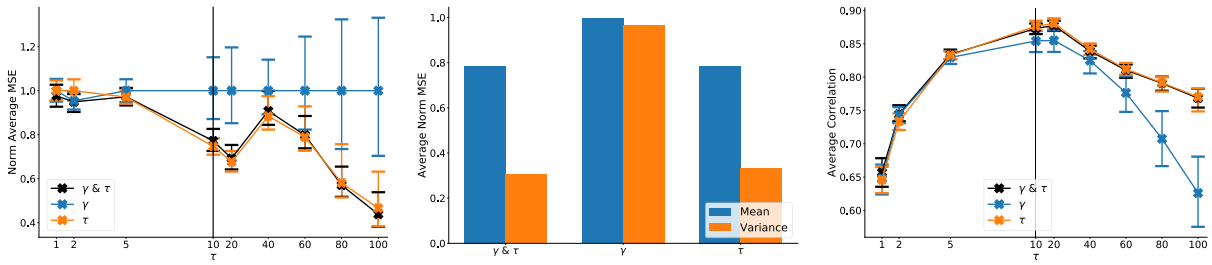


(a) MSE (b) Avg Norm MSE (c) Correlation (d) Learning Curves

Figure 7.15: **Atari@200M: Embedding Comparison.** We find no consistent difference between using the *direct* or *l_embed* approaches.

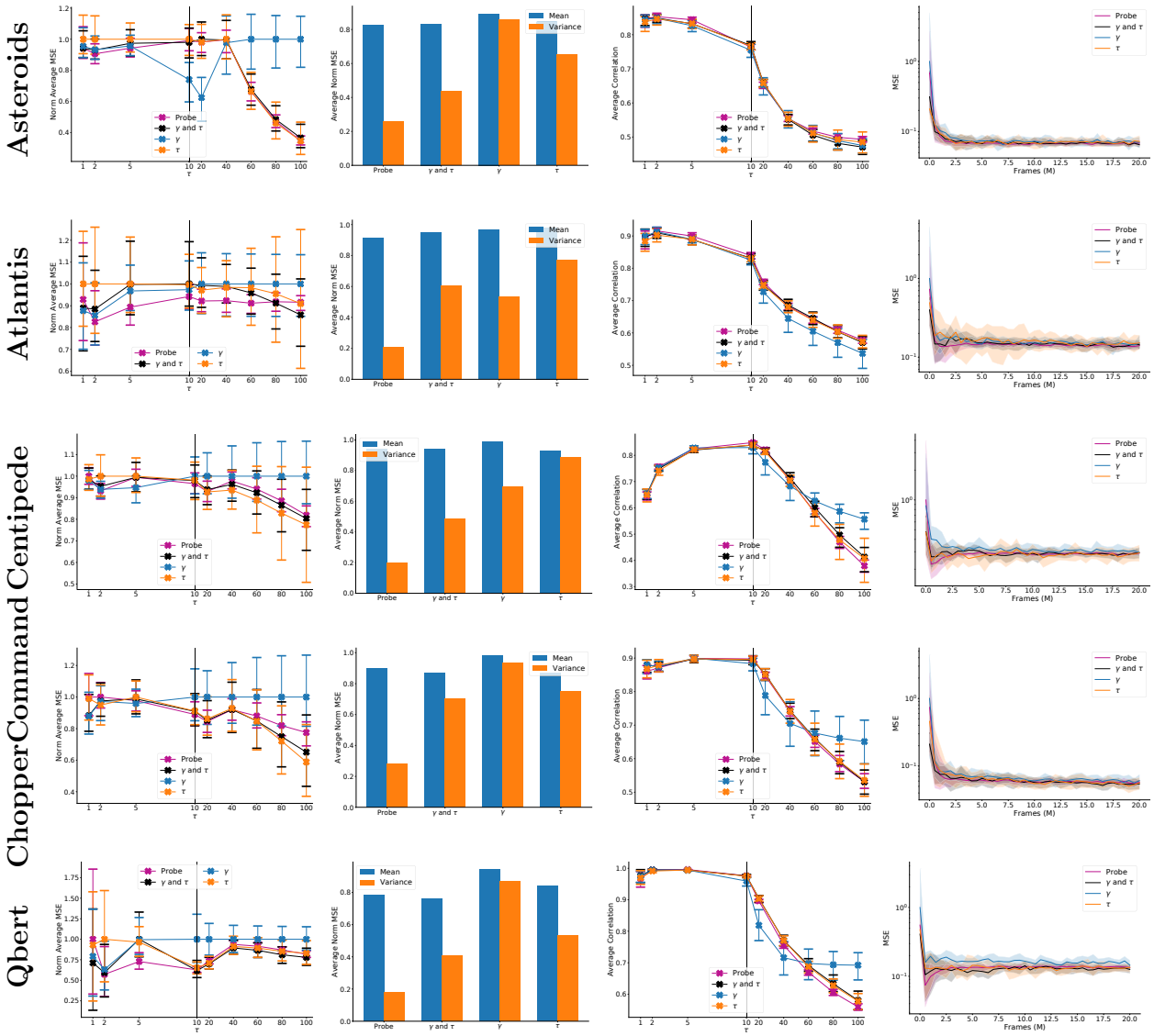
Timescale Input Comparison

We examine how the input timescale representation affects prediction performance (Figures 7.16, 7.17). We consider whether to use γ or τ inputs or both. The γ input values are naturally scaled between $[0, 1)$ and the τ input values were normalized by dividing by the max τ, which in these experiments was 100. We consistently see that using only γ produced the worst performance (Asteroids, in Figure 7.17, is an exception). Providing τ or both τ and γ performed very similarly, but we consistently observed that providing both representations performed best for very short timescales and had lower variance.



(a) MSE (b) Avg Norm MSE (c) Correlation

Figure 7.16: **Input Comparison.** We compare performance of the network when providing γ, τ or both to the network inputs. We see that providing only γ as the input timescale does the worst. Providing both γ and τ or just τ perform similarly, but providing both does better at very short timescales.

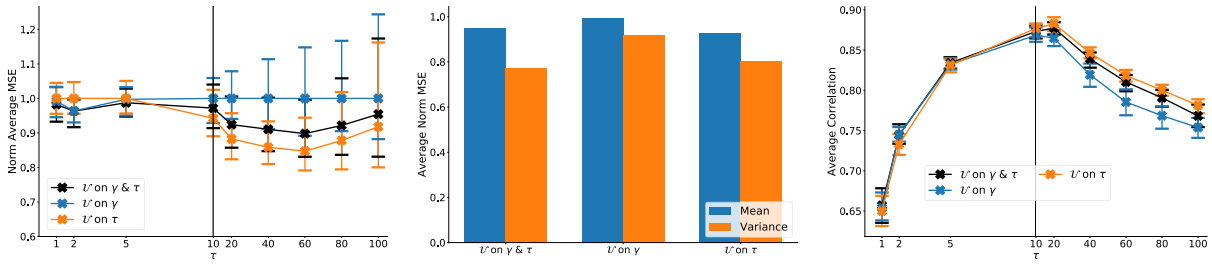


(a) MSE (b) Avg Norm MSE (c) Correlation (d) Learning Curves

Figure 7.17: **Atari@200M: Inputs Comparison.** We see that using γ as input gives better results at very short timescales than τ, but otherwise τ is better. Overall, providing both γ and τ provides the best performance. Although, the results in Asteroids are notable exception.

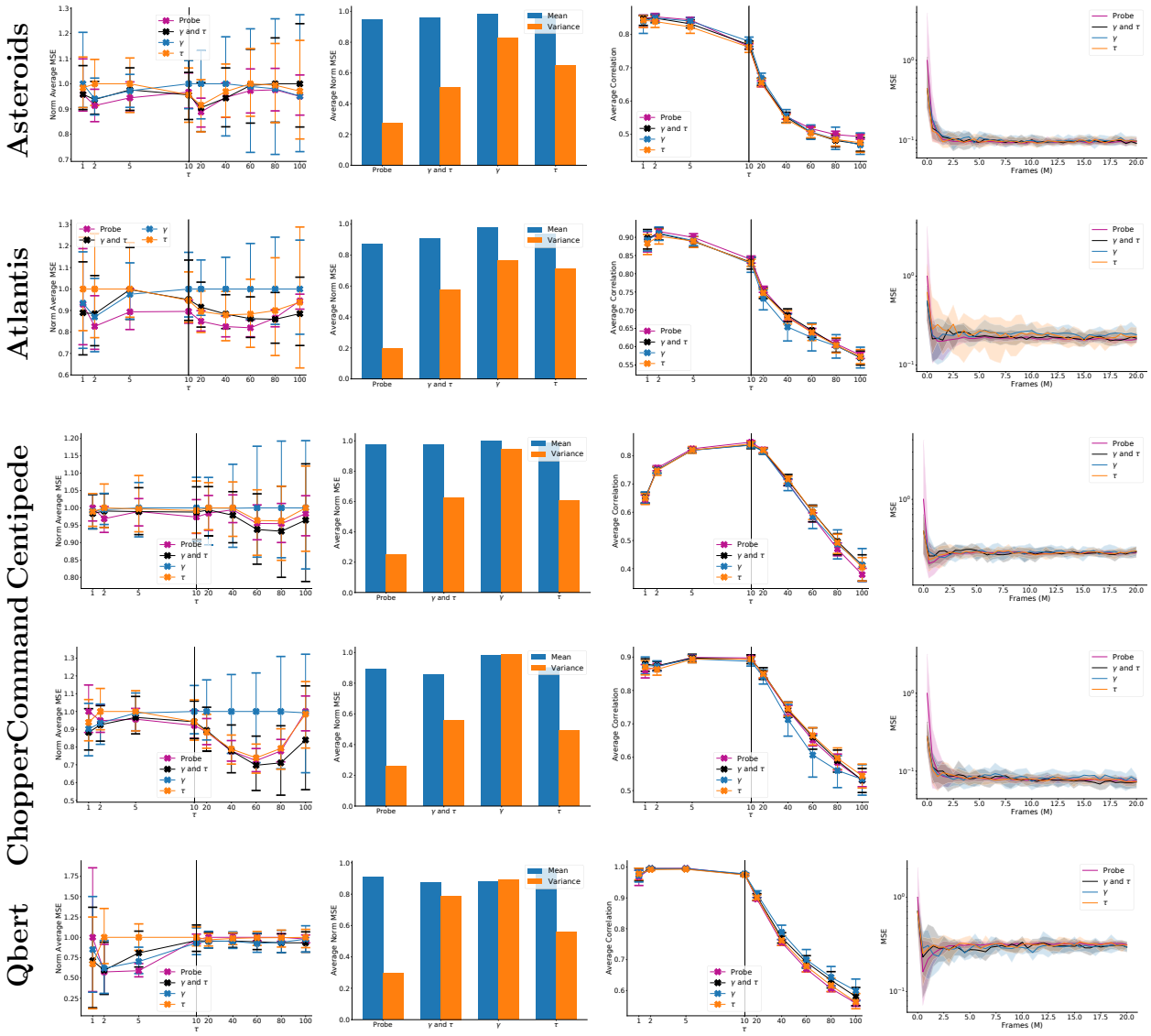
Distribution Comparison

We look at the effect of drawing Γ_t from different distributions (Figures 7.18, 7.19). We use a Γ_t of size 8, two of which are always the lower and upper bounds $\tau = \{1, 100\}$. Six additional γ_k are drawn from a given distribution. We either draw all six uniformly from the γ or τ scale or draw half from each. We see that drawing solely from γ performs worst overall, particularly at longer timescales, as is expected. Surprisingly, γ did not consistently outperform τ at very short timescales. If we consider all timescales and games evaluated there is no clear winner between drawing solely from τ or from τ and γ. However, at very short timescales drawing from both tended to produce better results. Thus, we recommend drawing from both scales as a default.



(a) MSE (b) Avg Norm MSE (c) Correlation

Figure 7.18: **Distribution Comparison.** We compare different distributions used to generate Γ_t. At lower timescales sampling from the γ scale does better than sampling from the τ scale and the opposite holds at longer timescales. Sampling from both provides a compromise in performance.

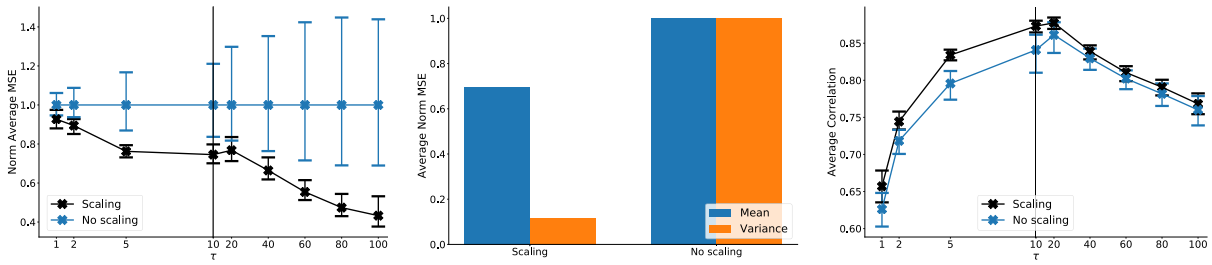


(a) MSE (b) Avg Norm MSE (c) Correlation (d) Learning Curves

Figure 7.19: **Atari@200M: Distribution Comparison.** Populating Γ_t from the τ scale is better than from γ scale except at very short timescales. Drawing samples from both scales does best overall.

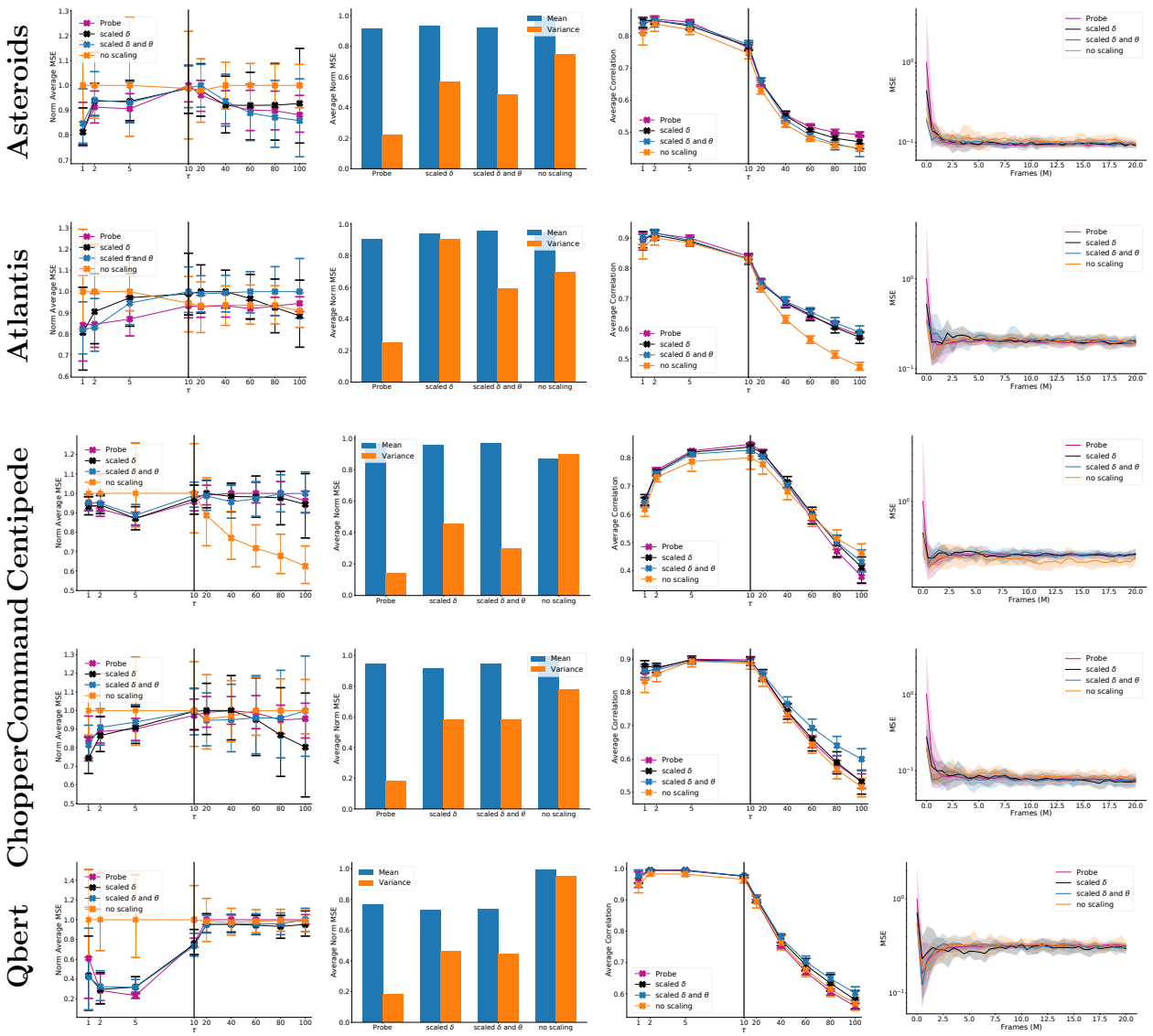
Loss Scaling

We examined the effect that loss scaling has on network performance. Figure 7.20 shows that on Centipede@25M there is a clear benefit, with clearly lower MSE and variance. Scaling the loss was expected to improve short timescale performance. Surprisingly, in terms of MSE, the greatest impact was on longer timescales. However, such a pronounced difference was not seen in other Atari games (Figure 7.21). Instead we saw a general trend in which scaling did improve performance at short scales at the cost of performance at mid and long timescales, which was in line with our expectations (again, Asteroids was somewhat an exception).



(a) MSE (b) Avg Norm MSE (c) Correlation

Figure 7.20: **Scaling Comparison.** We examined the effects that scaling the loss by $(1-\gamma_k)$ has. We see that scaling results in lower overall error and variance. Note that such a clear separation was not observed over other games tested.

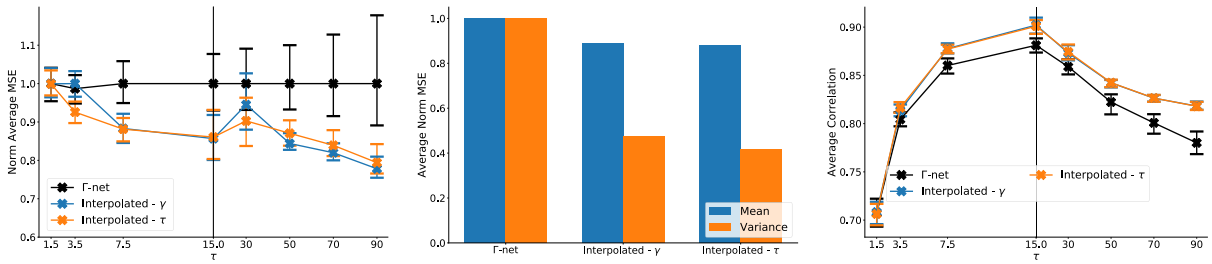


(a) MSE (b) Avg Norm MSE (c) Correlation (d) Learning Curves

Figure 7.21: **Atari@200M: Scaling Comparison.** As expected scaling the loss generally helped improve performance for shorter timescales, at the cost of performance elsewhere. We note that the error at initialization, which can be see in the learning rate plots, can be much lower without the scaling. This is due to the division by $(1-\gamma)$ used in the scaling networks, which has the effect of amplifying the noise in the initialization. To counter this we tried reducing the initial network weights, θ, by multiplying by $(1-\gamma)$ (*scaled* δ *and* θ). This did improve the initial error and matched the loss scaling for performance. This also appeared to reduce variance in some cases.

Estimation by Interpolation

An alternative approach to estimating value at arbitrary timescales is to have multiple prediction heads, each at a fixed timescale, and then linearly interpolate between the nearest bracketing timescales. In Figure 7.22 we show results for such an interpolation. Here we took the previously trained probe networks (with scaled loss and the taper network architecture) and performed linear interpolation for $\tau = [1.5, 3.5, 7.5, 15, 30, 50, 70, 90]$. As examples, to predict value for $\tau = 1.5$ we linearly interpreted predictions from the probe networks for the timescales $\tau = \{1, 2\}$ and for predictions at $\tau = 70$ we interpolated with predictions from $\tau = \{60, 80\}$ probe networks. Because of the non-linear relationship between τ and γ the linear interpolation gives different weighting depending on whether the interpolation is done on the τ or γ scale. Interpolating in these spaces is also compared. Results show that performance was fairly similar between the interpolation scales, but that the Γ-net did not perform as well. While it might have been expected that the ability of the neural network used by Γ-net to capture the non-linearity of the timescales would give it an advantage, this was not shown in this experiment. Rather, we suspect that the increased accuracy of the probe networks allowed the interpolation approach to win out.



(a) MSE (b) Avg Norm MSE (c) Correlation

Figure 7.22: **Interpolated.** Predictions are made between the probe timescales by taking the weighted average of the predictions made by the bracketing probe networks. Due to the non-linear relationships of γ and τ different weightings are produced if the weighting is done in either scale. We see that either interpolation produces better results than the Γ-nets.

7.6 Discussion

We have empirically evaluated various approaches to constructing Γ-nets and compared their predictive accuracy to baseline predictors. While we sought to separate the impacts of the various approaches, in reality *all of the variants we explored performed similarly.* We have

considered several different Atari games with deep learning architectures as well as a robotics demonstration using a shallow architecture. Overall we found that Γ-nets worked reliably both for reward and sensorimotor prediction.

Despite the relatively minor differences in performance across the variants we do make some recommendations for implementation. Since there was no universal difference between the *direct* or *l_embed* embedding approaches we recommend just using the simplest, *direct*. If looking for a general approach that is not specifically adapted to the task then we recommend using both γ and τ as inputs to the network as well as drawing samples from both scales in order to populate Γ_t. On the other hand, if longer timescales are preferred then it seems sufficient to use only τ for both input and sampling distribution. With regards to scaling the loss a clear universal benefit has not been observed and we suggest that further investigation is required to determine the best way to balance the losses resulting from different timescales. Such an investigation is a clear opportunity for future work.

Our method is thus far limited to the fixed discounting case. However, one of the key generalizations of GVFs is to support transition-dependent discounting functions: $\gamma_{t+1} \equiv \gamma(S_t, A_t, S_{t+1})$ (White, 2017). This allows GVFs to be more expressive in terms of the types of returns they can estimate. Extending our method to support such discounting is clearly an important next-step in this work.

There are several ways in which our work and that of Fedus et al. (2019) are complimentary. First, they demonstrated that using value predictions at many different timescales could serve as useful auxiliary tasks for driving representation learning. A clear next step is to investigate whether or not Γ-nets could also serve as a useful auxiliary task. Additionally, they demonstrated that alternative returns, such as hyperbolic discounting, could be formed by using value estimates at multiple timescales as a basis function; Γ-nets could provide such a basis function using a single network.

Long timescale predictions can be difficult to learn due to the higher variance of the observed returns. Romoff et al. (2019) presented an algorithm which computes values for an ordered set of timescales by predicting the differences between the values using separate network heads. Value estimates are then constructed in a cascade fashion where each timescale prediction adds to the one that came before it. They showed that their method could improve estimation accuracy for longer timescales by leveraging the accuracy of the easier to learn shorter timescales. We might expect a similar effect using Γ-nets where long timescale

predictions could benefit from the short timescales being learned directly in the network. Our current evaluation approach is not fine grained enough to discern such a benefit. Thus, this area warrants further exploration.

Γ-nets is related to other works which seek to learn many different predictions simultaneously and tractably. The UVFA (Schaul, Horgan, et al., 2015), on which this work is based, generalizes over goals. The successor representation (SR) (Dayan, 1993) separates environment dynamics from reward, providing a way to transfer learning across tasks (Barreto, Dabney, et al., 2017; Sherstan, Machado, and Pilarski, 2018). These ideas have been combined (Mankowitz et al., 2018; Ma et al., 2018) to enable transfer learning over multiple goals using off-policy learning. However, these methods still use fixed timescales, thus, a natural extension of Γ-nets is to combine them with these approaches.

The original motivation for this work was to use Γ-nets to create GVFs which form a predictive representations of state for use by the agent's policy. It now seems that the best approach would be to use multiple heads with predictions at fixed timescales and let the policy network learn to generalize over those predictions as it needed. Such an approach could be costly in terms of network weights and Γ-nets might accomplish the same thing with a smaller network.

7.7 Conclusion

We presented Γ-nets, a simple technique for generalizing value estimation across timescale. This technique allows a system to make predictions for values of any timescale within the training regime of the network. We expect that this ability will be useful in areas such as predictive representations of state—i.e., modelling the world as a collection of predictions about future sensorimotor signals. In complex environments complete models are not feasible, thus, being able to query for predicted outcomes at any timescale makes a model potentially more compact and expressive. An investigation of Γ-nets in different control learning scenarios is an important area for future work, and we believe they may be of benefit to ongoing research in planning and lifelong learning. In particular Γ-nets are complimentary to approaches which seek to learn many things about the world simultaneously such as the successor representation and universal value functions, suggesting that Γ-nets may provide us with a functional new tool for the pursuit of knowledgeable intelligent systems.

Part III

Driving Representation Learning with GVFs

In the third and final part of this dissertation we consider how GVFs can be used to drive representation learning. GVFs can be used as auxiliary tasks whose gradients assist in training a core representational network that is shared with an agent's policy. Auxiliary tasks have been shown to improve agent learning in many different works. In Section 8.2 we look at the use of Γ-nets as auxiliary tasks and in Section 8.3 we experimentally test the effect that the prediction timescale of a GVF-based auxiliary task has on performance.

Chapter 8

GVFs as Auxiliary Tasks

Thus far, we have considered two aspects of representation and GVFs. Part I looked at representation with GVFs and Part II looked at how GVF answers might themselves be represented and learned. A third consideration is how GVFs can be used to drive representation learning, which is the focus of this chapter. We present two investigations on the topic. In Section 8.2 we show experimentally that Γ-nets, introduced in Chapter 7, can improve policy learning in Atari games when used as an auxiliary task. In Section 8.3 we ask the question of how the timescale parameter of GVF-based auxiliary tasks affects policy learning and present experimental results.

8.1 Introduction

In reinforcement learning (RL), an agent tries to learn how to act so as to maximize the amount of reward it will receive over the rest of its lifetime. RL agents are dependent on their representations of state — their description, in vector form, used to describe the configuration of its external and internal environment.

In end-to-end training of deep reinforcement learning (DRL) the agent's representation, policy, and value estimates are trained simultaneously from reward received in the environment. Despite the successes in DRL, representation learning remains a serious challenge. A recent line of research is to use *auxiliary tasks* to aid in driving the optimization process (Jaderberg et al., 2017; Fedus et al., 2019; Le et al., 2018; Bellemare, Dabney, Dadashi, et al., 2019; Hernandez-Leal et al., 2019; Kartal et al., 2019; Veeriah et al., 2019). Let us define a task as any output of the network for which a loss function is attached. A *primary task* is then any task that is directly related to an agent improving its policy. In value-

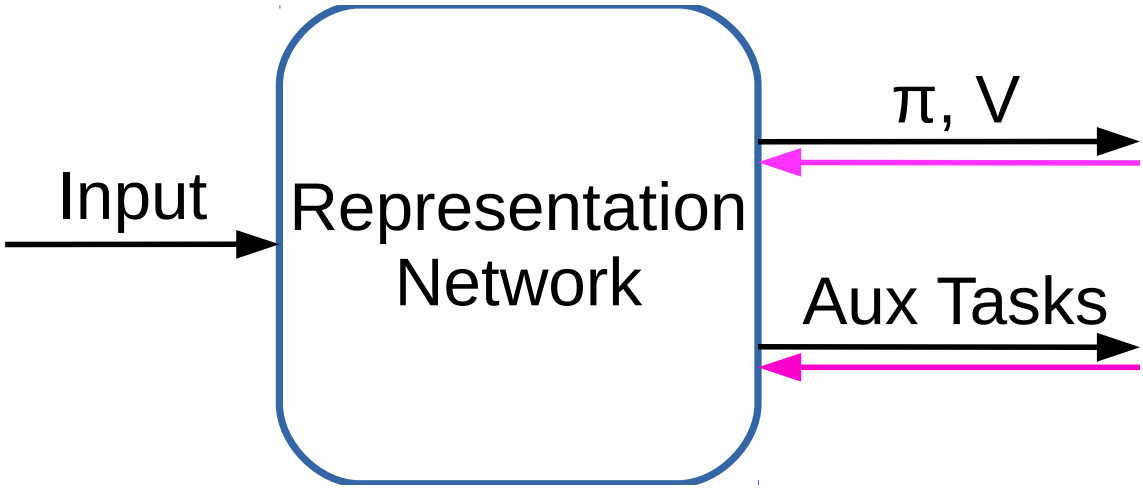


Figure 8.1: Auxiliary Tasks. Gradients (pink) flow back from both the primary tasks (π and V) and the auxiliary tasks. In this dissertation, we assume that the output of the auxiliary tasks serve no purpose except to compute losses and provide gradients.

based methods, this would include the value estimate, while in policy gradient methods this might include policy, value, and entropy losses (used to prevent early collapse of the policy). While what qualifies as an auxiliary task is open to interpretation, for the purposes of this dissertation let us define them as any task whose sole purpose is to assist in driving representation learning by providing additional gradients. These ideas are illustrated in Figure 8.1 where both the primary tasks and auxiliary tasks depend on the same shared representation network and provide gradients for training the value-based reinforcement learner's network.

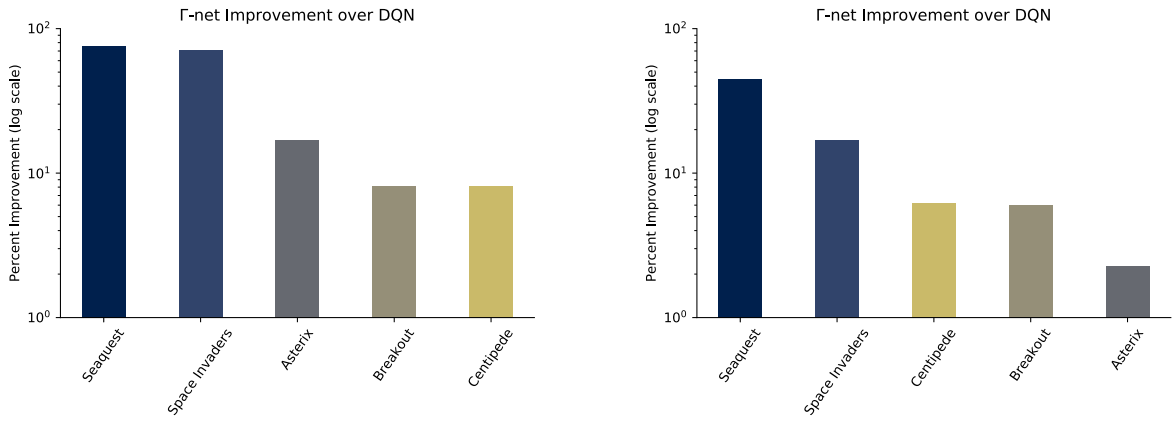
In this chapter we say that one representation is better than another if it enables either: 1) better final performance of the policy or 2) faster learning of the same policy. In this context, auxiliary tasks have been shown to both help and hinder performance (Fedus et al., 2019). At present, it is not well understood what tasks make good auxiliary tasks or why they help in general.

One example, where auxiliary tasks might help, is in the sparse reward setting, in which informative reward is infrequently observed. Auxiliary tasks may help by providing denser gradients. For example, an autoencoder auxiliary task (Jaderberg et al., 2017) can provide training gradients on every training sample. Let us distinguish two cases then. In one case, the auxiliary task helps the network find the same representations as it would otherwise find, but faster. This may occur because the auxiliary task gradients drive the network towards the good representation faster or because they make the representation easier to learn (e.g., the

additional gradients have the effect of reducing variance in the updates or make it easier to escape local minima (Suddarth and Kergosien, 1990)). In another case, these dense gradients impact the optimization process such that the representation found is in fact different, and possibly better. While we do not consider this setting here, a representation might also be considered "better" if it produces features that are more disentangled and generalize better in the transfer learning setting (Le et al., 2018). One of the results observed by Jaderberg et al. (2017) was that auxiliary tasks could make policy learning more robust to variations in learning parameters. We report on a similar effect in Section 8.3.

In this chapter we present two related works which investigate how GVFs as auxiliary tasks can affect policy learning. An oft suggested use for Γ-nets, introduced in Chapter 7, is as auxiliary tasks. In Section 8.2 we show that Γ-nets can indeed improve policy learning when used in this way. The Γ-nets training algorithm trains against a wide range of timescales. A natural question to ask is whether or not there is a relationship between the auxiliary task prediction timescale and performance. This question is explored in Section 8.3.

8.2 Gamma-nets as Auxiliary Tasks



(a) (b)

Figure 8.2: Percent improvement over DQN calculated by where score is the mean over **a)** the last 100 training episodes, **b)** all time.

It has often been suggested that Γ-nets would be useful as auxiliary tasks. In fact, related work by Fedus et al. (2019) suggests that predictions at multiple timescales provides useful auxiliary tasks. In their work multiple predictive heads hang off a single network and each

head predicts value at a different, fixed timescale. They evaluated performance across the entire ALE (Bellemare, Naddaf, et al., 2015) suite of games and saw clear improvements on some games as well as negative impacts on others. While we do not make a comparison to this approach we do present results on five ALE games using Γ-nets as auxiliary tasks. *In all of the games we examined we saw improved policy performance.*

To isolate the effect Γ-nets have on performance we train our agent using the standard DQN setup (Mnih, Kavukcuoglu, et al., 2015) rather than the Rainbow algorithm used in Chapter 7. Specifically, the changes added by Rainbow, which are not added here, are n-step prediction targets, prioritized replay sampling, and distributional value function representation. The core network consists of three convolutional layers with $\{32, 64, 64\}$ filters and a kernel size of $\{8, 4, 4\}$ respectively. This is then followed by a fully connected layer of 512 nodes with ReLU activation. The output of this layer is considered the output of the shared core network, which we refer to as ϕ. DQN uses Q-learning, that is, it predicts the state-action values for each state and action pair. Our policy network takes $\phi(s)$ as input and passes it through a single fully-connected layer with linear activation. This layer consists of $|\mathcal{A}|$ nodes, one corresponding to each $Q(s, a) \forall a \in \mathcal{A}$. Like in Section 7.5.3 we provide both γ and τ representations as inputs to the Γ-net. That is, for a given timescale, say γ_k, we compute τ_k and then concatenate both as inputs. The timescales are then linearly embedded by passing through a fully-connected layer of size 16 with linear activation. The output of the linear timescale embedding is concatenated with the output of the core network (a total of 528 features) to serve as the input to the Γ-net. The Γ-net network consists of a fully-connected layer of 256 nodes with ReLU activation and finally a fully-connected layer with linear activation which makes the predictive output, $V(S_t, \gamma_k)$. To train the Γ-net we sample eight timescale each from both the γ and τ scales for a total of 16 training timescales. Each training update of the entire augmented DQN network consisted of sampling 32 transitions from the replay buffer. Thus, for each update the effective batch size of the Γ-net is 512 (32 transitions * 16 timescales).

When combining auxiliary tasks with the core tasks it is typical to apply a scaling factor to each loss. Intuitively this is done to adjust the relative importance of each task in the optimization process. Practically, this is treated as an algorithm meta-parameter which is tuned by parameter sweeps. For all core tasks we use a scaling of 1 and for the Γ-net we apply a scaling β, which was swept over $\beta \in \{0.01, 0.1, 0.5\}$ on the game Space Invaders. For

Game	DQN		DQN + Γ-net	
Space Invaders	1003.29	(99.40)	1710.38	(207.87)
Seaquest	2304.38	(784.24)	4047.37	(1617.34)
Breakout	114.81	(26.53)	124.18	(29.70)
Centipede	3399.53	(202.95)	3673.02	(167.78)
Asterix	2531.70	(234.22)	2961.30	(298.46)

Table 8.1: Performance over the last 100 training episodes after 200M frames for DQN and DQN + Γ-net. Results are averaged over 10 seeds and standard deviation between runs is reported in parentheses.

each value of β we carried out 5 runs of 200M frames of experience and determined $\beta = 0.1$ performed best based on the final 100 episodes. Loss for both the core value network and the Γ-net used the Huber loss, which uses the squared loss for magnitudes below 1 and the absolute loss for magnitudes above. Unlike in Section 7.4 we did not applying loss scaling to the Γ-net.

Each agent is trained for 200M frames on five Atari 2600 games: Asterix, Breakout, Centipede, Space Invaders, and Seaquest. We follow the experimental methodology outlined in Machado, Bellemare, Talvitie, et al. (2018). The specific hyperparameters used are listed in Table 8.2. The performance of each agent was measured over the last 100 episodes of training from 200M frames of agent experience. Each experiment was further averaged over 10 distinct seeds.

Like Fedus et al. (2019) we saw improvements in the learned policies on several Atari games (Figure 8.2). In all the games we tested, average performance was better with Γ-nets as auxiliary tasks, whether considering final performance or performance over the entire run (Table 8.1). Figure 8.3 shows the average learning curves. In some cases the difference in performance is apparent from early on, while in others it is a gradual improvement. Here too it is clear that the auxiliary tasks helped performance. All results are averaged over 10 seeds.

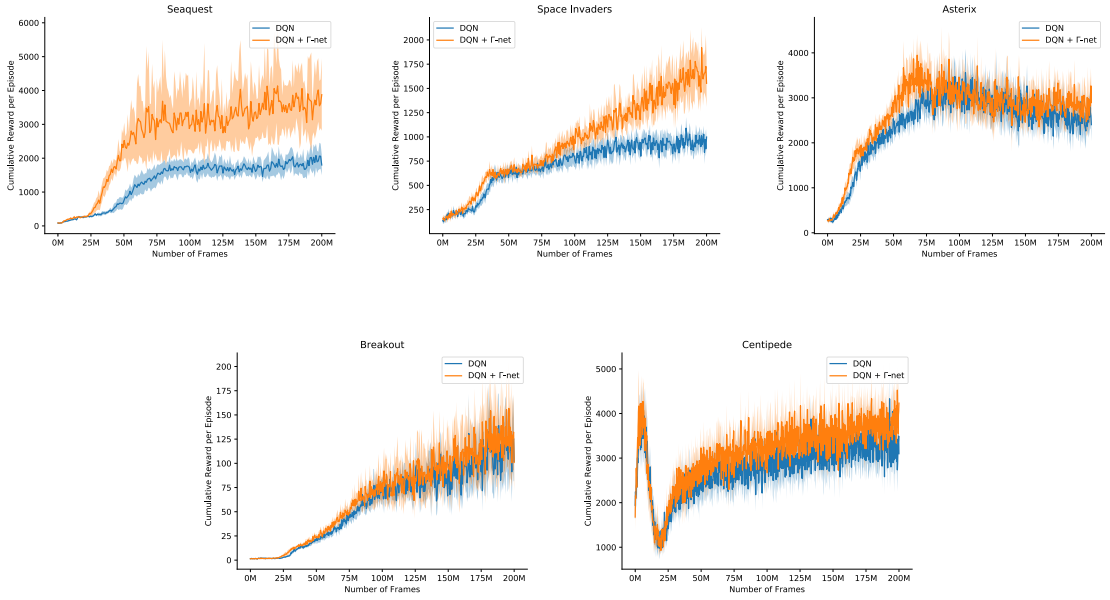


Figure 8.3: Training curves for DQN baseline and DQN + Γ-net. The x-axis is the number of frames during training. The y-axis is cumulative reward. Shading indicates 95% confidence intervals.

Table 8.2: Hyperparameters for Aux-task Γ-net experiments on the Arcade Learning Environment

DQN Parameters	
Frame stack	4
Frame skip	5
Replay buffer size	$1,000,000$
Replay buffer sampling	Uniform
Target update frequency	4
Batch size	32
Optimizer	RMSProp
RMSProp ρ	0.95
RMSProp ϵ	0.01
ϵ decay horizon	1M frames
ϵ initial	1.0
ϵ final	0.01
DQN γ	0.99
Sticky action probability	0.25

Γ-net Parameters	
Loss scaling β	0.1
γ sampling interval	$\gamma \in [0, 0.99)$
Γ batch size	16
Embedding size	32
Network layers	FC 256 $\rightarrow$ ReLU $\rightarrow$ FC 1

8.3 The Effect of Prediction Timescale on Representation Learning

Predictive auxiliary tasks have been shown to improve performance in numerous reinforcement learning works, however, this effect is still not well understood. The primary purpose of this section is to investigate the impact that an auxiliary task's prediction timescale has on the agent's policy performance. We consider auxiliary tasks which learn to make on-policy predictions using temporal-difference learning. We test the impact of prediction timescale using a specific form of auxiliary task in which the input image is used as the prediction target, which we refer to as temporal-difference autoencoders (TD-AE). We empirically evaluate the effect of TD-AE on the A2C algorithm in the ViZDoom environment using different prediction timescales. While we do not observe a clear relationship between the prediction timescale on performance, we make the following observations: 1) using auxiliary tasks allows us to reduce the trajectory length of the A2C algorithm, 2) in some cases temporally extended TD-AE performs better than a straight autoencoder, 3) performance with auxiliary tasks is sensitive to the weight placed on the auxiliary loss, 4) despite this sensitivity, auxiliary tasks improved performance without extensive hyper-parameter tuning. Our overall conclusions are that TD-AE increases the robustness of the A2C algorithm to the trajectory length and while promising, further study is required to fully understand the relationship between auxiliary task prediction timescale and the agent's performance.

8.3.1 Hypothesis

Numerous works have included predictive auxiliary tasks (Jaderberg et al., 2017; Veeriah et al., 2019; Fedus et al., 2019; Hernandez-Leal et al., 2019; Kartal et al., 2019). However, the relationship between the prediction timescale and policy performance has not been studied. Our hypothesis is that having auxiliary tasks that predict the future on extended timescales will enable the agent to learn better policies. Further, we hypothesize that as the prediction timescale increases, from reconstruction towards the policy timescale of the agent, we should expect improved policy performance. This is motivated by the belief that on-policy temporally extended auxiliary tasks will share similar or even the same feature dependencies as the policy.

8.3.2 Contributions

To test our hypothesis we use temporally extended predictions of the agent's input image as auxiliary tasks, which we refer to as temporal-difference autoencoders (TD-AE). We evaluate the impact of these auxiliary tasks on three scenarios in ViZDoom, a 3D game engine, using the A2C algorithm (Mnih, Badia, et al., 2016). Our experiments do not conclusively support or disprove our hypothesis, however, we make several observations. Our first, and most notable observation, is that using TD-AE makes the learning more robust to the length of the trajectories, n, used in training. Reducing n should allow the agent to adapt its policy more quickly and allows us to approach a more online algorithm. However, reducing n from 128 to 8 steps causes the baseline algorithm to fail. By including TD-AE auxiliary tasks the trajectory length can be reduced to 8 steps and still achieve performance which meets or exceeds that of the baseline. Our second observation is that, while we do not observe a direct relationship between auxiliary task timescale and policy performance, we do observe instances where the use of predictive auxiliary tasks outperform reconstructive tasks (i.e. TD-AE($\gamma = 0$)). Our third observation is that the performance with auxiliary tasks is highly sensitive to the weighting placed on their losses. Finally, despite this sensitivity, the TD-AE auxiliary tasks are shown to generally improve performance even when the weighting is not well-tuned.

8.3.3 Background

In RL, an agent learns to maximize its lifetime reward by interacting with its environment. This is commonly modelled as a Markov Decision Process (MDP) defined by $\mathcal{S}, \mathcal{A}, P, r, \gamma$, where $\mathcal{S}$ is the set of states, P describes the probability of transitioning from one state, s, to the next, s', given an action, a, from the set of actions $\mathcal{A}$, that is $P(s'|s, a)$. The reward function generates a reward as a function of the transition: $r \doteq r(s, a, s')$. Finally, $\gamma \in [0, 1]$ is a discount term applied to future rewards.

The agent's goal is to maximize the sum of undiscounted future rewards ($\gamma = 1.0$), known as the *return*. In practice we often use the discounted return instead:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots. \tag{8.1}$$

Measuring the goodness of being in a particular state and acting according to policy π

is given by the *state-value*—the expected return from the state:

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t \middle| S_t = s\right]. \tag{8.2}$$

The goodness of taking an action from a state and then following policy π, known as the *state-action value*, is given as:

$$q_\pi(s,a) \doteq \mathbb{E}\pi\left[G_t \middle| S_t = s, A_t = a\right]. \tag{8.3}$$

Temporal-difference (TD) methods (Sutton and Barto, 2018) update a value estimate, V, towards samples of the discounted bootstrapped return by computing the TD-error as:

$$\delta_t = R_{t+1} + \gamma V_\pi(S_{t+1}) - V_\pi(S_t). \tag{8.4}$$

An n-step TD-error, which is used in A2C, uses n samples of the return before bootstrapping:

$$\delta_t = \left(\sum_{i=1}^{n} \gamma^{i-1} R_{t+i}\right) + \gamma^n V_\pi(S_{t+n}) - V_\pi(S_t). \tag{8.5}$$

Sutton, Modayil, et al. (2011) broadened the use of value estimation by introducing general value functions (GVFs). GVFs make two changes to the value function formulation. The first is to replace reward with any other measurable signal, which is now referred to as the *cumulant*, C. The second change is to make discounting a function of the transition, $\gamma_t \equiv \gamma(S_t, A_t, S_{t+1})$ (White, 2017). Thus, the value function now estimates the expectation of the following return:

$$G_t = C_{t+1} + \gamma_{t+1} C_{t+2} + \gamma_{t+1}\gamma_{t+2} C_{t+3} + \dots.$$

GVFs model elements of an agent's sensorimotor stream as temporally extended predictions. They can be learned using standard temporal-difference learning methods (Sutton and Barto, 2018).

Advantage actor-critic (A2C) (Mnih, Badia, et al., 2016) is an algorithm that employs a parallelized synchronous training scheme (e.g., using multiple CPUs) for efficiency; it is an on-policy RL method that does not use an experience replay buffer. In A2C multiple agents simultaneously accumulate transitions using the same policy in separate copies of the environment. When all workers have gathered a fixed number of transitions the resulting batch of samples is used to train the policy and value estimates. A2C maintains a parameterized policy (actor) $\pi(a|s;\theta_\pi)$ and value function (critic) $V(s;\theta_v)$, parameterized by θ_π and

θ_v respectively. The value estimate is updated using the n-step TD-error (Eq. (8.5)). It is common to use a softmax output layer for the policy head $\pi(A_t|S_t;\theta_\pi)$ and one linear output for the value function head $V(s_t;\theta_v)$, with all non-output layers shared (see Figure 8.1). The loss function of A2C is composed of three terms: policy loss, $\mathcal{L}_\pi$, value loss, $\mathcal{L}_v$ and negative entropy loss, $\mathcal{L}_H$. Entropy is treated as a bonus to discourage the policy from prematurely converging; the policy is rewarded for having high entropy. Each of the losses is weighted by a corresponding scalar, λ. Thus, the A2C loss is:

$$\mathcal{L}_{A2C} = \mathcal{L}_\pi + \lambda_v \mathcal{L}_v + \lambda_H \mathcal{L}_H,$$

where each loss component is weighted by a scalar λ with $\lambda_v = 0.5$, $\lambda_H = 0.001$.

8.3.4 Temporally Extended Auxiliary Tasks

Consider writing value in the following form:

$$\begin{aligned} V_\pi(s) =& \mathbb{E}[R_{t+1}] + \gamma \mathbb{E}[R_{t+2}] + \gamma^2 \mathbb{E}[R_{t+3}] + \dots \\ &+ \gamma^{T-t-1} \mathbb{E}[R_T] + \gamma^{T-t} V_\pi(S_T), \end{aligned}$$

where each expectation is conditioned on $S_t = s$ and taken over the policy π. The value function does not directly observe state s, but instead uses a feature vector ϕ to describe the state, thus, $\phi_t \equiv \phi(S_t)$. Let us further say that to predict the expected reward k steps in the future requires some feature vector $\phi(S_t;k)$, which contains all information required. That is, to accurately predict $\mathbb{E}[R_{t+k}|S_t = s]$ requires all the information represented by $\phi(S_t;k)$. While ϕ might be independent for all k, $V_\pi(s)$ depends on the entire set of ϕ for all k. Thus, for a given cumulant and policy, value estimates at all timescales depend on the same features and the gradients produced during optimization should ideally lead to representations which capture all the same feature dependencies as required by the policy.

However, the discounting term γ places more emphasis on near term rewards (for all $\gamma < 1.0$), thus the features for near term rewards have a stronger impact on the value estimates performance. This dependency is further modulated by the magnitude of the rewards at each timestep in the future. Thus, the optimization will focus on some of these features over the others. Therefore, given that our RL agents care about reward off into the

future—typical values of γ are $0.9, 0.95, 0.99$—it is reasonable to think that auxiliary tasks that share the same temporal feature dependencies would drive the network towards the same representation.

8.3.5 Methods

To study the effects of the auxiliary task temporal prediction length on representation learning, we first need to define prediction targets. Here we simply choose the image input, X_t, as the prediction target, giving us the GVF defined as:

$$\mathbf{\Psi}_\pi(s) = \mathbb{E}_\pi\Big[X_t + \gamma \mathbf{\Psi}_\pi(S_{t+1}) | S_t = s\Big]. \tag{8.6}$$

This requires a small change in the GVF definition. While the cumulant is usually defined as a function of the transition, $C_{t+1}(S_t, A_t, S_{t+1})$, here, instead, the cumulant is only dependent on the start of the transition and is thus subscripted by t (similar to the form used for defining the successor representation (Sherstan, Machado, and Pilarski, 2018)). This GVF, which we refer to as TD-AE(γ), has the form of a temporally extended autoencoder. Note that when $\gamma = 0$ the target is simply the reconstruction of the input, i.e., an autoencoder. This GVF can be learned using standard TD methods and throughout the experiments in this section we use TD(0), where the loss is:

$$\delta_t^2 = (X_t + \gamma \mathbf{\Psi}_\pi(S_{t+1}) - \mathbf{\Psi}_\pi(S_t))^2.$$

TD-AE is a conceptually unusual prediction target. When the input is an image, the sum of future discounted images has no obvious interpretation. However, we can instead think of each pixel in the image as a GVF cumulant, in which case the TD-AE is a collection of a large number of GVFs with well defined target signals. Each GVF predicts the sum of discounted future values for a certain pixel. In the GVF setting this is straightforward and does not require an easily interpreted description.

Sherstan, Dohare, et al. (2020) proposed a method for scaling TD losses by their timescale such that losses and corresponding gradients for different timescale value estimates are approximately of the same magnitude. We use this scaling here, which involves two modifications. The first is to rescale the network output by explicitly dividing by $(1 - \gamma)$. The

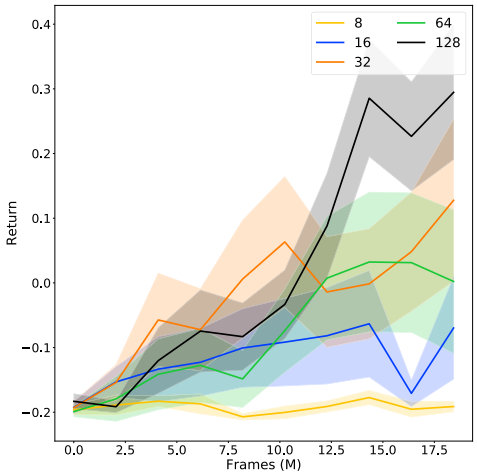


Figure 8.4: The effect of sequence length on performance on the K-Item 2 scenario in ViZDoom. Legend indicates the length of the sequence length used for training. As the sequence length reduces the baseline becomes increasingly likely to fall into a failure mode.

second is to multiply the TD-error by the same factor. An average loss is taken across all d pixels of the image. Thus, in our experiments, the loss used for the TD-AE is computed as follows:

$$\mathcal{L}_{TD-AE} = \frac{1}{d}\sum_{i}^{d}[(1-\gamma)(X_{t:i} + \gamma \mathbf{\Psi}_{\pi;i}(S_{t+1}) - \mathbf{\Psi}_{\pi;i}(S_t))]^2.$$

Combining this loss with the A2C loss gives:

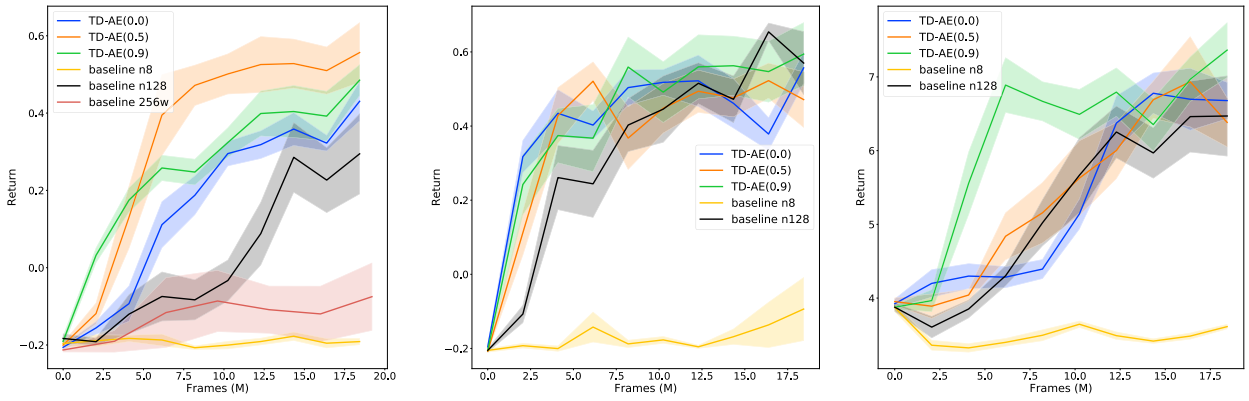
$$\mathcal{L} = \mathcal{L}_{A2C} + \lambda_{TD-AE}\mathcal{L}_{TD-AE}$$

Where λ_{TD-AE} is a loss scaling parameter we sweep over in our experiments.

8.3.6 Experiments

Setting

ViZDoom (Kempka et al., 2016) is an open source implementation of the Doom video game. This is a customizable 3D first-person environment. For evaluation in the ViZDoom environment, we used the A2C base code from Beeching et al. (2019). In our experiments, 16 agents run simultaneously collecting transition samples that are used for batch training of the agent. The baseline uses a fairly long update sequence: 128 steps. Thus, each training batch consists of 2048 transition tuples obtained from 16 sequences of 128 steps. The agent's policy's discount was $\gamma = 0.99$.



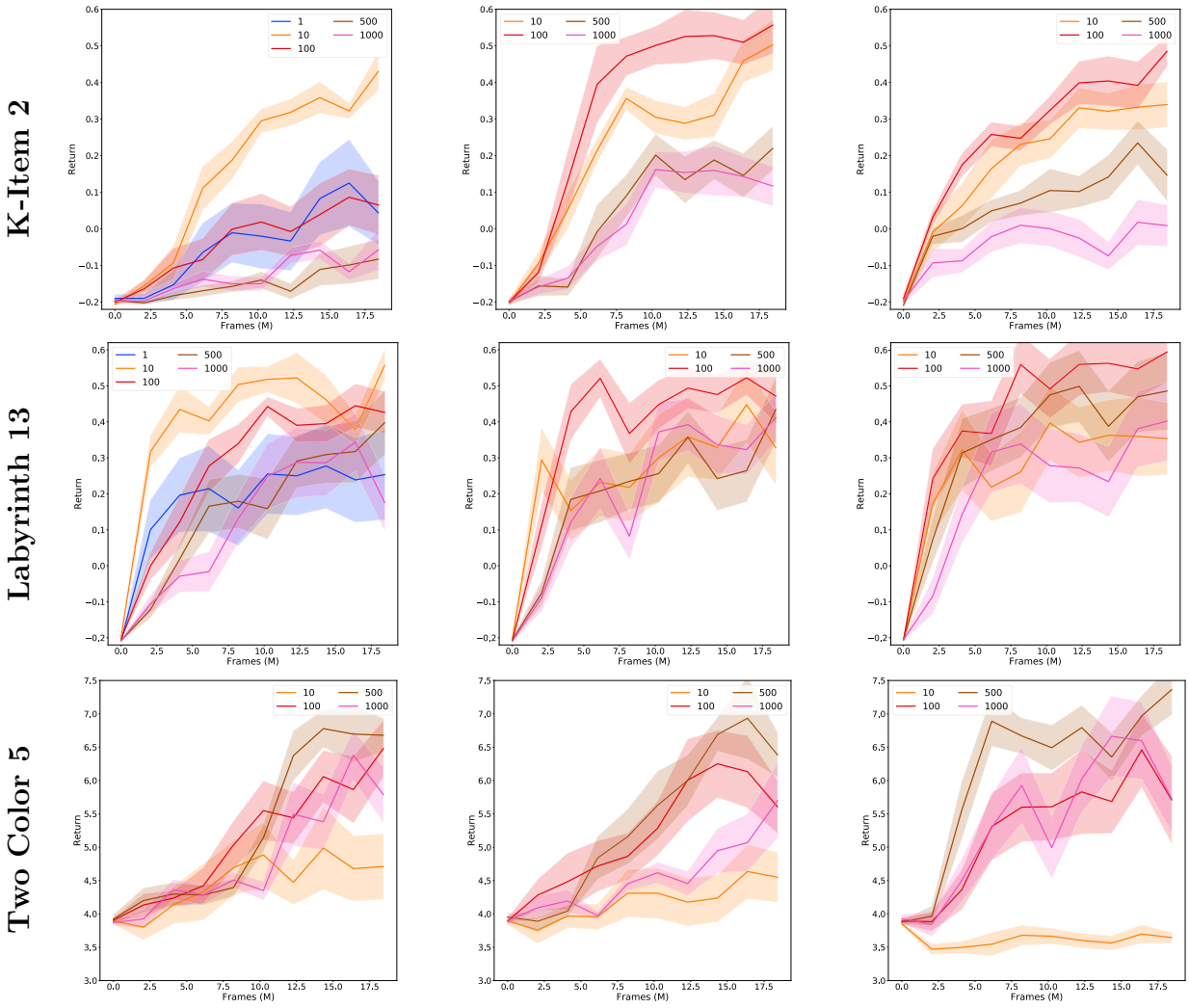
(a) K-Item 2 (b) Labyrinth 13 (c) Two Color 5

Figure 8.5: Performance with and without TD-AE auxiliary task on three ViZDoom scenarios. The original baseline with n=128 is given in black. The baseline with n=8 is given in yellow and we see that it collapses and is not able to learn. Adding the TD-AE restores the performance of the network and in some cases even outperforms the original baseline. For the performance of the various TD-AE we chose the best λ_{TD-AE} from our sweeps (Table 8.3).

The core network consists of 3 convolutional layers followed by a fully connected layer with each layer using ReLU activation. This is fed into a GRU memory layer whose output is used as input to the policy, value, and TD-AE heads. The decoder for the TD-AE is composed of 3 fully connected layers with sigmoid activations. The sizes of the first two layers are 256 and 512 and the final layer, which has a linear activation, outputs the same size and shape as the original input images (64×112 pixels with 3 color channels).

Each experiment was run over 10 seeds for 20 million frames (using 16 CPUs $\approx$ 10 hrs per run). Our figures use shading to indicate the standard error. While 10 seeds is not enough to give statistical certainty of the mean, we use standard error here for display purposes. Thus, shading should be considered as a scaled indication of the variability about the mean. Evaluations for the plots were performed by intermittently running test phases (approximately every 2 M frames). For each test phase 50 games were run with frozen network weights and the average episode return was reported.

Multiple ViZDoom scenarios were investigated, including *K-Item 2, Labyrinth 13*, and *Two Color Maze 5* (see descriptions in Beeching et al. (2019)). These scenarios were chosen as they required memory of the past and were thus expected to have long-term dependencies.



(a) TD-AE(0.0) (b) TD-AE(0.5) (c) TD-AE(0.9)

Figure 8.6: Effect of auxiliary loss weighting.

Reducing Sequence Length n

We observe a consistent behavior in all of the tasks: when we reduce the length of the update sequence performance suffers (Figure 8.4). However, adding an auxiliary task can restore baseline performance and in some cases exceed it. Figure 8.5 shows this behavior across different ViZDoom scenarios. The default performance, with no auxiliary tasks and a sequence length of 128, is listed as *baseline n128*. In all tasks decreasing the sequence length to $n = 8$, *baseline n8*, significantly reduces performance. Adding in the TD-AE consistently restores performance, and in some cases exceeds the performance observed on *baseline n128*.

Table 8.3: Values of λ_{TD-AE} swept over. Optimal value for each γ is indicated in bold. The size of these weights are much larger than reported in other works; this is due to the loss scaling of $1 - \gamma$ we described in Section 8.3.5

γ	λ_{TD-AE}
K-Item 2 and **Labyrinth 13**	
0.0	{1, **10**, 100, 500, 1000}
0.5, 0.9	{10, **100**, 500, 1000}
Two Color 5	
0.0, 0.5, 0.9	{10, 100, **500**, 1000}

The performance reported for each series uses the best λ_{TD-AE} found in our sweep. While we do not see a clear relationship between the timescale of the TD-AE prediction we do observe that in two of the scenarios using a $\gamma > 0$ outperforms an autoencoder, $\gamma = 0.0$ (Figures 8.5a, 8.5c).

Reducing the sequence length has the effect of reducing the batch size. In Figure 8.5a we show an additional experiment, *baseline 256w*, which used no auxiliary tasks, $n = 8$, and 256 workers, producing the same batch size as *baseline n128*. Simply restoring the batch size did not restore the performance.

Sensitivity to Loss Weighting

For each scenario, we swept over λ_{TD-AE} (see Table 8.3) and found the performance of the policy could be sensitive to this term. This is shown in Figure 8.6 which depicts performance for different γ and weighting. Too little or too much weighting reduced performance. This is consistent with results found by others (Hernandez-Leal et al., 2019). However, we note that in some works the same weighting is used for all scenarios. This could be problematic as our results showed that the same weighting was not optimal in all our scenarios. We might speculate on the reason for this sensitivity as follows. If there is too little weight the resulting gradients are too small and result in smaller update steps, making it slower to learn the policy and decreasing the likelihood of escaping local minima or failure modes. On the other hand, if there is too much weight then the gradients of the auxiliary losses might interfere with those of the policy (Schaul, Borsa, et al., 2019).

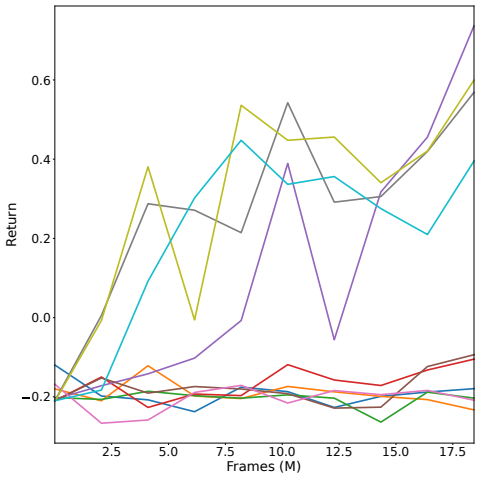


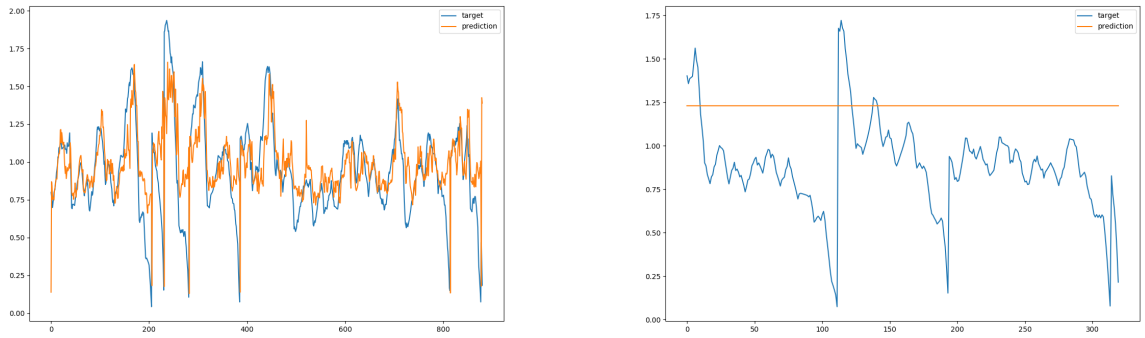
Figure 8.7: The mean and standard error can hide the fact that the returns split into two different modes (K-Item2, baseline n32). Each series indicates a different random seed. Here six of ten seeds are in the failure mode.

Bi-Modal Performance

In the plots we have shown the mean of multiple runs along with the standard error. However, this hides the underlying behavior. The performance of the agent would generally fall into two distinct modes. In the first mode, which we'll refer to as the *learning* mode, the performance curve continues to improve with more training, for example *baseline n128*. In the second mode, which we refer to as the *failure* mode, learning quickly plateaus and generally does not seem to recover, for example *baseline n8*. Figure 8.4 suggests that as n decreases, the likelihood of achieving the learning mode decreases. Thus, presenting the mean masks the underlying distribution of returns like those shown in Figure 8.7, which shows the performance of each seed for *baseline n32*. We see that six out of the ten seeds resulted in the failure mode. Thus, the mean underestimates those times when the policy learning succeeds and overestimates the failure mode.

TD-AE Predictions

Figure 8.8 considers auxiliary predictions of two different pixels. Transitions were gathered sequentially starting from an episode reset and potentially spanning several task terminations and resets. The true return is shown in blue and the predictions of TD-AE(0.9) are given in orange. While some predictions matched the target well, Figure 8.8a, others failed and were not even predictive, such as the pixel shown in Figure 8.8b which is a constant value.



(a) (b)

Figure 8.8: Prediction quality for individual pixels for TD-AE(0.9) (Two Color 3). For some pixels the prediction (orange) matches the target (blue) very well as in (a), while for other pixels the prediction does very poorly as in (b).

8.3.7 Discussion and Future Work

We set out to test the hypothesis that temporally extended auxiliary tasks improve policy learning. We expected that as the timescale of the auxiliary task became longer, improvement would increase. If this hypothesis held true it would give some direction on the kinds of auxiliary tasks to use in a learning system.

We observed that even autoencoders (TD-AE(0.0)), a commonly used auxiliary task (Le et al., 2018; Shelhamer et al., 2016), significantly improved learning. Further, in some cases, we saw that TD-AE with $\gamma > 0$ had a greater effect on performance. However, we did not observe any trend where larger γ consistently performed better. Thus, additional study is required to validate or disprove our hypothesis. There are several directions in which this work could proceed. The first is to consider other domains, particularly settings with longer time dependencies. It may be that in the settings we studied, all of the relevant information could be captured by short timescale dependencies and longer ones added no additional information. Also, studying our hypothesis with the architecture in this section may have been complicated by the use of the GRU memory unit that was employed. It may be clearer to study simpler architectures which lack explicit memory structures.

Instead, our primary contribution is the observation that adding auxiliary tasks allows us to shorten the trajectory length used in the A2C update. Shortening the trajectories may be useful for enabling the agent to update its policy more frequently, adapting quicker

to changes in the environment. We believe this represents a general trend that auxiliary tasks can make policy learning less sensitive to parameter settings. This sort of reduced sensitivity, or increased robustness, was previously reported by Jaderberg et al. (2017); our work supports that observation.

One of the unanswered questions from our experiments is why does learning fail when the trajectory length is shortened? Generally, as the trajectory length was shortened, performance degraded. However, it did not degrade smoothly, but instead appeared to fall into failure modes where, after a short period of learning, the policy was unable to improve further.

Policy performance can be sensitive to the weighting placed on the auxiliary tasks. Further, the optimal weighting is not the same for all games. This points out a potential problem in works that select a fixed weighting by sweeping across a subset of environments and then use the same weighting across all experiments thereafter. Consider that in the worst case, where an auxiliary task would interfere with the learning of the representation needed by the policy, the optimal weighting would be zero. Thus, it seems that any auxiliary task, with a properly tuned weighting, should, at worst, have no effect on policy performance and otherwise improve it. Therefore, reporting negative impacts on performance by an auxiliary task does not necessarily provide information about the usefulness of the auxiliary task, but rather is likely an indicator that the weighting is poorly tuned. Thus, it would be beneficial to use an algorithm that automatically tunes the weighting parameters, such as the meta-RL algorithm employed by Xu et al. (2018).

We close by restating our observations: 1) adding TD-AE tasks can improve the robustness of the A2C algorithm to its trajectory length parameter setting, 2) temporally extended TD-AE can sometimes help more than just an autoencoder, 3) performance of the policy is sensitive to the weighting placed on its loss, and 4) even poorly tuned weightings enhanced agent performance.

8.4 Discovering GVF Auxiliary Tasks

This chapter would be incomplete without reporting on recent work by Veeriah et al. (2019). In their work GVF auxiliary tasks are learned in an end-to-end manner using a meta-gradient approach. Their system contains two deep neural networks. The first is the standard *core*

network, which consists of a shared representation network off of which the policy, value, and auxiliary tasks hang. The second network, the *question* network, learns to output cumulants and discounts as a function of the agent's observations. The outputs of this network then define the GVFs auxiliary tasks of the core network, which are learned in an on-policy manner along with the usual actor-critic updates. For details on how these networks are trained the interested reader should see the paper.

Their experiments demonstrated that, on their own, without gradients from the primary tasks, the learned auxiliary GVFs were sufficient to drive representation learning; their agent achieved comparable performance to the baseline agent which was trained with gradients from the primary task. Further, by combining gradients from both the primary and auxiliary tasks the agent was able to outperform the baseline in some scenarios. Perhaps the most interesting contribution of their paper is that it demonstrates an approach in which GVFs can be discovered directly from data, without designer intervention.

8.5 Conclusion

This chapter presented investigations into using GVFs as auxiliary tasks for driving representation learning. In Section 8.2 we demonstrated that Γ-nets could be used effectively as an auxiliary task, improving policy performance in the five Atari games that were tested. In Section 8.3 we sought to understand the effect that the prediction timescale of an auxiliary task has on the representation, as measured by changes in policy performance. While we did not see any clear relationship between the prediction timescale and policy performance it appears that generally predictive auxiliary tasks may outperform strictly reconstructive ones such as the autoencoder. However, more research is required to confirm this. Additionally, an important observation is that adding the auxiliary tasks increased the robustness of the learning algorithm, allowing us to reduce the batch trajectory lengths. This result, combined with those reported by Jaderberg et al. (2017) suggest that a major benefit of auxiliary tasks may be to reduce the sensitivity of our learning algorithms to parameter settings. This may be a valuable tool towards creating more general learning systems.

These works were clearly preliminary and more research is required to understand auxiliary tasks in general. As well, the question we posed in Section 8.3, what is the relationship between auxiliary task prediction timescale and representation learning, remains unanswered.

Chapter 9

Conclusion

My dissertation has focused on the relationship between general value functions (GVFs) and representation. Capturing knowledge as a GVF consists of two parts: 1) the predictive question 2) the estimated answer; this dissertation contributes understanding to both the question and the answer. In Part I, several contributions to GVF questions were presented. In Chapter 3, I proposed that internally generated signals should be candidates for prediction targets, just as external sensorimotor signals are. These introspective measures and predictions might then be used to enhance an agent's state representation allowing the agent to learn general policies to control its own learning process. In Chapter 4, I presented several empirical investigations of different introspective measures and predictions that suggested ways in which they might be used to enhance an agent's decision making abilities. Chapter 5 then presented a new method for directly estimating the variance of the return by predicting the squared TD-error of a value learner. This showed how a GVF could be used to predict statistical properties of the agent's environment conditioned on its own policy. Part II provided several contributions to how GVF questions can be answered. Chapter 6 demonstrated that using the successor representation to represent GVFs allows incrementally added GVFs to be learned faster by leveraging existing knowledge about the dynamics of the agent's environment. Chapter 7 demonstrated Γ-nets, a method for generalization GVF estimation over timescale by including timescale as part of the function approximator's inputs, allowing a single estimator to make predictions for any fixed timescale within its training regime. In Chapter 8, I presented investigations on how GVFs could be used as auxiliary tasks for driving representation learning and thereby improving policy learning. Section 8.2 showed that Γ-nets could improve policy learning in the deep reinforcement

learning setting on Atari when used as auxiliary tasks. Finally, Section 8.3 examined the effect that prediction timescale of GVF auxiliary tasks has on policy learning. While we were unable to find a clear relationship between the GVF timescale and policy performance we did observe that GVF auxiliary tasks helped make the learning algorithm more robust.

The majority of contemporary research in RL focuses on evaluations on single tasks and compares algorithms by their expected returns. In contrast, the work presented in this dissertation is motivated, if not evaluated, in the continual learning (CL) setting. In CL an agent that learns *adequate* competencies that are transferable across many settings is preferred to an agent that can *optimally* solve exactly one task. The works presented in this dissertation represent small pieces to solving this bigger puzzle, but they are not meant to be taken on their own; they should be viewed as part of a larger system.

Modelling the world is not a new idea in AI, nor are predictions, in fact these problems are fundamental. What sets GVFs apart is that they provide a way to ground a predictive syntax in a well-known formulation—the value function. Indeed, with only three components: a policy, a cumulant, and a continuation function, we are able to describe a wide variety of predictive questions—questions that are grounded in an agent's own experience and representation. Further, TD algorithms provide a computationally efficient and well known way for learning GVF answers. Thus, the appeal of GVFs is that they provide two things: 1) an expressive formalism capable of describing a great deal of world knowledge that is grounded in the agent's own sensorimotor experience, 2) known computational methods for efficiently learning estimates of GVF answers.

As of the writing of this dissertation there remain many open questions in GVF research. One of the biggest is how to use them. Should they be a state-representation in and of themselves or is it enough to have them drive learning of a state-representation as auxiliary tasks? How can they be used in planning? The majority of existing works, including those presented here, have focused on demonstrating that they can be learned or showing how to improve their learning or expressiveness. What remains to be seen is a clear demonstration that they are in fact useful. While unsatisfying from the perspective of research in CL, one of the clearest uses for GVFs is in conjunction with hand-crafted policies such as the demonstrations of Pavlovian control referenced in Chapter 2. To summarize, the idea is that prescripted control policies are triggered when a particular GVF prediction crosses a threshold. These sorts of predictions have been used effectively in a number of situations

and have a strong basis in animal reflexes. They have been used for learning to avoid motor over-current situations in autonomous mobile robots (Modayil and Sutton, 2014). Dalrymple et al. (2020) used GVF predictions to trigger gait transitions in a state-machine used to produce walking by electrically stimulating an animal's spinal cord. Daftry et al. (2016) used GVF like predictions to trigger safety procedures in autonomous drones. It seems natural that these ideas could readily be carried forward to other situations, even those deployed in real-world settings.

In this dissertation I have motivated work on GVFs as part of an agent that incrementally constructs its own predictive model of the world over time. However, with very few exceptions, there has been little work on GVF *discovery*—the creation of new GVFs over time. This is a major hurdle to validating the premise of GVFs. There are two main approaches to this work. The first is a generate-and-test approach that randomly generates GVFs and then later evaluates if they are useful (Schlegel, White, and White, 2018). Several experiments have shown that random generation can produce useful representations (Sutton and Whitehead, 1993; Mahmood and Sutton, 2013). However, this approach ignores the wealth of data that exists in the agent's sensorimotor stream. In addition to random generation, or possibly instead of it, I believe that GVF generation should include a data driven method. The only known demonstration of data-driven GVF generation is that of Veeriah et al. (2019). They use a meta-gradient method to train a network to output cumulants and discounts for on-policy GVFs. These GVFs are then used as auxiliary tasks for driving representation learning in the core network. These auxiliary tasks were shown to improve learning of the primary task. This work demonstrated that useful GVFs could be learned directly from data. GVF generation, or discovery, represents an under-explored opportunity for research. However, the reason it is under-explored is because it is difficult. The work presented in this dissertation may contribute to GVF generation by suggesting ways in which the GVFs should be represented. For example, by factoring GVF representation with the SR, as done in Chapter 6, we can clearly separate policy and cumulant discovery. Further, the application of question embeddings, as was done with the timescale parameter in Chapter 7, may provide a general mechanism for specifying GVF parameters and learning generalized estimates.

GVFs in their current form have been studied for nearly a decade, however, the underlying motivations for their use have yet to be validated. While this might be discouraging it is not

an indicator that the approach is misguided or fruitless. GVFs are attempting to address one of the fundamentally difficult problems of AI—how does an agent construct a representation of the world from its own experience. This is not a trivial problem and has itself been studied for many more decades. I continue to believe that the study of GVFs and predictive models in general is a worthy research path and one that will ultimately pay off. I anticipate numerous advances in this field in the next decade and believe that the work presented in this dissertation will assist in their discovery.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Yangqing, J., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Retrieved from https://www.tensorflow.org/

Barreto, A., Borsa, D., Quan, J., Schaul, T., Silver, D., Hessel, M., Mankowitz, D., Žídek, A., & Munos, R. (2018). Transfer in Deep Reinforcement Learning Using Successor Features and Generalised Policy Improvement. In *International Conference on Machine Learning (ICML)*. Stockholm, Sweden.

Barreto, A., Dabney, W., Munos, R., Hunt, J., Schaul, T., Silver, D., & van Hasselt, H. (2017). Transfer in Reinforcement Learning with Successor Features and Generalised Policy Improvement. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 4055–4065). Long Beach, USA.

Barto, A. G. (2013). Intrinsic Motivation and Reinforcement Learning. In G. Baldassarre & M. Mirolli (Eds.), *Intrinsically Motivated Learning in Natural and Artificial Systems* (pp. 17–48). Springer.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., & Petersen, S. (2016). DeepMind Lab. *arXiv*, *1612.03801*.

Beeching, E., Wolf, C., Dibangoye, J., & Simonin, O. (2019). Deep Reinforcement Learning on a Budget: 3D Control and Reasoning Without a Supercomputer. *arXiv*, *1904.01806*.

Bellemare, M. G., Dabney, W., & Munos, R. (2017). A Distributional Perspective on Reinforcement Learning. In *International Conference on Machine Learning (ICML)* (pp. 449–458). Sydney, Australia.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2015). The Arcade Learning Environment: An Evaluation Platform for General Agents. In *International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 4148–4152). Lille, France.

Bellemare, M. G., Dabney, W., Dadashi, R., Taïga, A. A., Castro, P. S., Le Roux, N., Schuurmans, D., Lattimore, T., & Lyle, C. (2019). A Geometric Perspective on Optimal Representations for Reinforcement Learning. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 4358–4369). Vancouver, Canada.

Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., & Munos, R. (2016). Unifying Count-Based Exploration and Intrinsic Motivation. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 1471–1479). Barcelona, Spain.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation Learning : A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828.

Bengio, Y. & Delalleau, O. (2011). On the Expressive Power of Deep Architectures. In *International Conference on Algorithmic Learning Theory (ALT)* (pp. 18–36). Espoo, Finland.

Bionic Limbs for Improved Natural Control (BLINC). (2020). Retrieved from https://blinclab.ca/

Bridges, M. M., Para, M. P., & Mashner, M. J. (2011). Control System Architecture for the Modular Prosthetic Limb. *Johns Hopkins APL Technical Digest*, *30*(3), 217–222.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., & Bellemare, M. G. (2018). Dopamine: A Research Framework for Deep Reinforcement Learning. Retrieved from http://arxiv.org/abs/1812.06110

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., & Zhang, Z. (2015). MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *arXiv*, *1512.01274*.

Daftry, S., Zeng, S., Bagnell, J. A., & Hebert, M. (2016). Introspective Perception: Learning to Predict Failures in Vision Systems. In *IEEE International Conference on Intelligent Robots and Systems (IROS)* (pp. 1743–1750). Daejeon, Korea.

Dalrymple, A. N., Roszko, D. A., Sutton, R. S., & Mushahwar, V. K. (2020). Pavlovian Control of Intraspinal Microstimulation to Produce Over-Ground Walking. *Journal of Neural Engineering*, *17*(3).

Dayan, P. (1993). Improving Generalization for Temporal Difference Learning: The Successor Representation. *Neural Computation*, *5*(4), 613–624.

De Asis, K., Bennett, B., & Sutton, R. S. (2018). Predicting Periodicity with Temporal Difference Learning. *arXiv*, *1809.07435*.

Deisenroth, M. P. & Rasmussen, C. E. (2011). PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *International Conference on Machine Learning (ICML* (pp. 465–472). Bellevue, Washington, USA.

Dimitrakakis, C. (2006). *Ensembles for Sequence Learning* (Doctoral dissertation, EPFL).

Drescher, G. L. (1991). *Made-up Minds: A Constructivist Approach to Artificial Intelligence*. MIT Press.

Duck Rabbit Illusion. (1892). *Fliegende Blätter*, *October 23*. Retrieved from https://en.wikipedia.org/wiki/Rabbit-duck_illusion

Edwards, A., Dawson, M., Hebert, J., Sherstan, C., Sutton, R., Chan, K., & Pilarski, P. (2016). Application of Real-time Machine Learning to Myoelectric Prosthesis Control: A Case Series in Adaptive Switching. *Prosthetics and Orthotics International*, *40*(5), 573–581.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., & Kavukcuoglu, K. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *In-*

ternational Conference on Machine Learning (ICML) (pp. 1407–1416). Stockholm, Sweden.

Fedus, W., Gelada, C., Bengio, Y., Bellemare, M. G., & Larochelle, H. (2019). Hyperbolic Discounting and Learning over Multiple Horizons. *arXiv*, *1902.06865*.

Gehring, C. A. (2015). Approximate Linear Successor Representation. In *Reinforcement Learning Decision Making (RLDM)*. Edmonton, Canada.

Gehring, C. & Precup, D. (2013). Smart Exploration in Reinforcement Learning using Absolute Temporal Difference Errors. In *Autonomous Agents and Multiagent Systems (AAMAS)* (pp. 1037–1044). Saint Paul, Minnesota.

Gilbert, D. (2006). *Stumbling on Happiness*. Knopf.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Günther, J., Kearney, A., Dawson, M. R., Sherstan, C., & Pilarski, P. M. (2018). Predictions, Surprise, and Predictions of Surprise in General Value Function Architectures. In *AAAI Fall Symposium - Reasoning and Learning in Real-World Systems for Long-Term Autonomy* (pp. 22–29). Arlington, USA.

Hernandez-Leal, P., Kartal, B., & Taylor, M. E. (2019). Agent Modeling as Auxiliary Task for Deep Reinforcement Learning. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)* (Vol. 15, pp. 31–37). Atlanta, USA.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning. In *AAAI Conference on Artificial Intelligence (AAAI)* (pp. 3215–3222). New Orleans, USA.

Hu, H. & Kantor, G. (2017). Introspective Evaluation of Perception Performance for Parameter Tuning Without Ground Truth. In *Robotics: Science and Systems (RSS)*. Cambridge, USA.

Iwata, K., Ikeda, K., & Sakai, H. (2004). A New Criterion Using Information Gain for Action Selection Strategy in Reinforcement Learning. *IEEE Transactions on Neural Networks*, *15*(4), 792–799.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., & Kavukcuoglu, K. (2017). Reinforcement Learning with Unsupervised Auxiliary Tasks. In *International Conference on Learning Representations (ICLR)*. Vancouver, Canada.

Kahn, G., Villaflor, A., Pong, V., Abbeel, P., & Levine, S. (2017). Uncertainty-Aware Reinforcement Learning for Collision Avoidance. *arXiv*, *1702.01182*.

Kartal, B., Hernandez-Leal, P., & Taylor, M. E. (2019). Terminal Prediction as an Auxiliary Task for Deep Reinforcement Learning. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)* (Vol. 15, pp. 38–44). Atlanta, USA.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaśkowski, W. (2016). ViZDoom: A doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games (CIG)* (pp. 341–348). Santorini, Greece.

Kulkarni, T. D., Narasimhan, K. R., Saeedi, A., Tenenbaum, J. B., Saeedi CSAIL, A., & Tenenbaum BCS, J. B. (2016). Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. *Conference on Neural Information Processing Systems (NeurIPS)*, 3675–3683.

Le, L., Patterson, A., & White, M. (2018). Supervised autoencoders: Improving Generalization Performance with Unsupervised Regularizers. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 25–27). Montreal, Canada.

Lehnert, L., Tellex, S., & Littman, M. L. (2017). Advantages and Limitations of using Successor Features for Transfer in Reinforcement Learning. *arXiv*, *1708.00102*.

Littman, M., Sutton, R. S., & Singh, S. (2002). Predictive Representations of State. In *Conference on Neural Information Processing Systems (NeurIPS)* (Vol. 14, pp. 1555–1562). Vancouver, Canada.

Ma, C., Wen, J., & Bengio, Y. (2018). Universal Successor Representations for Transfer Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*. Vancouver, Canada.

Machado, M. C., Bellemare, M. G., & Bowling, M. (2020). Count-Based Exploration with the Successor Representation. In *AAAI Conference on Artificial Intelligence (AAAI)* (pp. 5125–5133). New York, USA.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., & Bowling, M. (2018). Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research*, *61*, 523–562.

Machado, M. C., Rosenbaum, C., Guo, X., Liu, M., Tesauro, G., & Campbell, M. (2018). Eigenoption Discovery Through The Deep Successor Representation. In *International Conference on Learning Representations (ICLR)*. Vancouver, Canada.

Maei, H. R. (2011). *Gradient Temporal-Difference Learning Algorithms* (Doctoral dissertation, University of Alberta).

Mahmood, A. R. (2017). *Incremental Off-policy Reinforcement Learning Algorithms* (Doctoral dissertation, University of Alberta).

Mahmood, A. R. & Sutton, R. S. (2013). Representation Search through Generate and Test. In *AAAI Workshop on Learning Rich Representations from Low-Level Sensors*. Bellevue, Washington, USA.

Makino, T. & Takagi, T. (2008). On-line Discovery of Temporal-Difference Networks. *International Conference on Machine Learning (ICML)*, 632–639.

Mankowitz, D. J., Žídek, A., Barreto, A., Horgan, D., Hessel, M., Quan, J., Oh, J., van Hasselt, H., Silver, D., & Schaul, T. (2018). Unicorn: Continual Learning with a Universal, Off-policy Agent. *arXiv*, *1802.08294*.

McCracken, P. & Bowling, M. (2006). Online Discovery and Learning of Predictive State Representations. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 875–882). Vancouver, Canada.

Mihalkova, L. & Mooney, R. (2006). Using Active Relocation to Aid Reinforcement Learning. In *FLAIRS Conference* (pp. 580–585). Melbourne Beach, USA.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)* (pp. 1928–1937). New York, USA.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015).

Human-level Control Through Deep Reinforcement Learning. *Nature*, *518*(7540), 529–533.

Modayil, J. & Sutton, R. S. (2014). Prediction Driven Behavior: Learning Predictions that Drive Fixed Responses. In *AAAI Workshop on AI and Robotics*. Quebec City, Canada.

Modayil, J., White, A., & Sutton, R. S. (2014). Multi-Timescale Nexting in a Reinforcement Learning Robot. *Adaptive Behavior*, *22*(2), 146–160.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., & Liang, E. (2004). Autonomous Inverted Helicopter Flight via Reinforcement Learning. In *International Symposium on Experimental Robotics (ISER)* (pp. 363–372). Singapore.

Ngo, H., Luciw, M., Förster, A., & Schmidhuber, J. (2013). Confidence-Based Progress-Driven Self-Generated Goals for Skill Acquisition in Developmental Robots. *Frontiers in Psychology*, *4*, 1–19.

Oudeyer, P. Y., Kaplan, F., & Hafner, V. V. (2007). Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE Transactions on Evolutionary Computation*, *11*(2), 265–286.

Oudeyer, P.-Y. & Kaplan, F. (2009). What is Intrinsic Motivation? A Typology of Computational Approaches. *Frontiers in Neurorobotics*, *1*, 1–14.

Parker, A. S., Edwards, A. L., & Pilarski, P. M. (2019). Exploring the Impact of Machine-Learned Predictions on Feedback from an Artificial Limb. In *IEEE International Conference on Rehabilitation Robotics (ICORR)* (pp. 1239–1246). Toronto, Canada.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 8024–8035). Vancouver, Canada.

Pilarski, P. M., Dick, T. B., & Sutton, R. S. (2013). Real-Time Prediction Learning for the Simultaneous Actuation of Multiple Prosthetic Joints. In *IEEE International Conference on Rehabilitation Robotics (ICORR)* (pp. 1–8). Seattle, USA.

Pilarski, P. M. & Sherstan, C. (2016). Steps Toward Knowledgeable Neuroprostheses. In *IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)* (p. 220). Singapore.

Pitis, S. (2018). Source Traces for Temporal Difference Learning. In *AAAI Conference on Artificial Intelligence (AAAI)* (pp. 3952–3959). New Orleans, Louisiana, USA.

Pouget, A., Drugowitsch, J., & Kepecs, A. (2016). Confidence and Certainty: Distinct Probabilistic Quantities for Different Goals. *Nature Neuroscience*, *19*(3), 366–374.

Prashanth, L. A. & Ghavamzadeh, M. (2013). Actor-Critic Algorithms for Risk-Sensitive MDPs. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 252–260). Lake Tahoe, USA.

Rafols, E. J., Ring, M. B., Sutton, R. S., & Tanner, B. (2005). Using Predictive Representations to Improve Generalization in Reinforcement Learning. In *International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 835–840). Edinburgh, Scotland.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Van de Wiele, T., Mnih, V., Heess, N., & Springenberg, T. (2018). Learning by Playing - Solving Sparse Reward Tasks from Scratch. *Proceedings of Machine Learning Research*, *80*, 4344–4353.

Ring, M. B. (1994). *Continual Learning in Reinforcement Environments* (Doctoral dissertation, The University of Texas at Austin).

Ring, M. B. (1997). CHILD: A First Step Towards Continual Learning. *Machine Learning*, *28*(1), 77–104.

Romoff, J., Henderson, P., Touati, A., Ollivier, Y., Brunskill, E., & Pineau, J. (2019). Separating Value Functions Across Time-scales. *Proceedings of Machine Learning Research*, *97*, 5468–5477.

Russek, E. M., Momennejad, I., Botvinick, M. M., Gershman, S. J., & Daw, N. D. (2017). Predictive Representations Can Link Model-based Reinforcement Learning to Model - free Mechanisms. *PLoS Computational Biology*, *13*(9), 1–42.

Sakaguchi, Y. & Takano, M. (2004). Reliability of Internal Prediction/Estimation and its Application. I. Adaptive Action Selection Reflecting Reliability of Value Function. *Neural Networks*, *17*(7), 935–952.

Sato, M., Kimura, H., & Kobayashi, S. (2001). TD Algorithm for the Variance of Return and Mean-Variance Reinforcement Learning. *Transactions of the Japanese Society for Artificial Intelligence*, *16*(3), 353–362.

Schaul, T., Borsa, D., Modayil, J., & Pascanu, R. (2019). Ray Interference: a Source of Plateaus in Deep Reinforcement Learning. *arXiv*, *1904.11455*.

Schaul, T., Horgan, D., Gregor, K., & Silver, D. (2015). Universal Value Function Approximators. In *International Conference on Machine Learning (ICML)* (pp. 1312–1320). Lille, France.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized Experience Replay. *arXiv*, *1511.05952*.

Schaul, T. & Ring, M. (2013). Better Generalization with Forecasts. In *International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1656–1662). Beijing, China.

Schlegel, M., White, A., Patterson, A., & White, M. (2018). General Value Function Networks. *arXiv*, *1807.06763*.

Schlegel, M., White, A., & White, M. (2018). A Baseline of Discovery for General Value Function Networks under Partial Observability. In *NeurIPS Workshop on Reinforcement Learning under Partial Observability):* Montreal, Canada.

Schmidhuber, J. (1991). Curious Model-Building Control Systems. In *IEEE International Joint Conference on Neural Networks (IJCNN)* (pp. 1458–1463). Singapore.

Shelhamer, E., Mahmoudieh, P., Argus, M., & Darrell, T. (2016). Loss is its Own Reward: Self-Supervision for Reinforcement Learning. *arXiv*, *1612.07307*.

Sherstan, C. (2015). *Towards Prosthetic Arms as Wearable Intelligent Robots* (Master's thesis, University of Alberta).

Sherstan, C., Ashley, D. R., Bennett, B., Young, K., White, A., White, M., & Sutton, R. S. (2018). Comparing Direct and Indirect Temporal-Difference Methods for Estimating the Variance of the Return. In *Conference on Uncertainty in Artificial Intelligence (UAI)* (pp. 63–72). Monterey, USA.

Sherstan, C., Dohare, S., MacGlashan, J., Günther, J., & Pilarski, P. (2020). Gamma Nets: Generalizing Value Estimation over Timescale. In *AAAI Conference on Artificial Intelligence* (pp. 5717–5725). New York, USA.

Sherstan, C., Kartal, B., Hernandez-Leal, P., & Taylor, M. E. (2020). Work in Progress: Temporally Extended Auxiliary Tasks. In *Adaptive and Learning Agents (ALA) Workshop at AAMAS*. Auckland, New Zealand (Virtual).

Sherstan, C., Machado, M. C., & Pilarski, P. M. (2018). Accelerating Learning in Constructive Predictive Frameworks with the Successor Representation. In *IEEE International Conference on Robots and Systems (IROS)* (pp. 2997–3003). Madrid, Spain.

Sherstan, C., Machado, M. C., Travnik, J., White, A., Vasan, G., & Pilarski, P. M. (2017). Confident Decision Making with General Value Functions. In *Reinforcement Learning and Decision Making (RLDM)*. Ann Arbor, USA.

Sherstan, C., Machado, M. C., White, A., & Pilarski, P. M. (2016). Introspective Agents: Confidence Measures for General Value Functions. In *Artificial General Intelligence (AGI)* (pp. 258–261). New York, New York, USA: Springer International Publishing.

Sherstan, C., Modayil, J., & Pilarski, P. M. (2015). A Collaborative Approach to the Simultaneous Multi-joint Control of a Prosthetic Arm. In *International Conference on Rehabilitation Robotics (ICORR)* (pp. 13–18). Singapore.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., & Degris, T. (2017). The Predictron: End-to-end Learning and Planning. In *International Conference on Machine Learning (ICML)* (pp. 3191–3199). Sydney, Australia.

Sobel, M. J. (1982). The Variance of Discounted Markov Decision Processes. *Journal of Applied Probability*, *19*(4), 794–802.

Stachenfeld, K. L., Botvinick, M. M., & Gershman, S. J. (2017). The Hippocampus as a Predictive Map. *Nature Neuroscience*, *20*, 1643–1653.

Storck, J., Hochreiter, S., & Schmidhuber, J. (1995). Reinforcement Driven Information Acquisition in Non-Deterministic Environments. *International Conference on Artificial Neural Networks (ICANN)*, *2*, 159–164.

Suddarth, S. C. & Kergosien, Y. (1990). Rule-Injection Hints as a Means of Improving Network Performance and Learning Time. In *Neural Networks* (pp. 120–129). Springer.

Sutton, R. S. & Whitehead, S. D. (1993). Online Learning with Random Representation. In *International Conference on Machine Learning (ICML)* (pp. 314–321). Amherst, MA, USA.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, *3*(1), 9–44.

Sutton, R. S. (1995). TD Models: Modeling the World at a Mixture of Time Scales. In *International Conference on Machine Learning (ICML)* (pp. 531–539). Tahoe City, USA.

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). The MIT Press.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, *112*(1), 181–211.

Sutton, R. S., Rafols, E. J., & Koop, A. (2006). Temporal Abstraction in Temporal-difference Networks. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 1313–1320). Vancouver, Canada.

Sutton, R. S., Maei, H., & Precup, D. (2009). Fast Gradient-Descent Methods for Temporal-Difference Learning with Linear Function Approximation. In *International Conference on Machine Learning (ICML)* (pp. 993–1000). Montreal, Canada.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., & Precup, D. (2011). Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (Vol. 2, pp. 761–768). Taipei, Taiwan.

Sutton, R. & Tanner, B. (2005). Temporal-Difference Networks. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 1377–1384). Vancouver, Canada.

Tamar, A., Castro, D. D., & Mannor, S. (2016). Learning the Variance of the Reward-To-Go. *Journal of Machine Learning Research*, *17*(13), 1–36.

Tamar, A., Di Castro, D., & Mannor, S. (2012). Policy Gradients with Variance Related Risk Criteria. In *International Conference on Machine Learning (ICML)* (pp. 1651–1658). Edinburgh, Scotland.

Tamar, A. & Mannor, S. (2013). Variance Adjusted Actor Critic Algorithms. *arXiv*, *1310.3697*.

Tanaka, S. C., Doya, K., Okada, G., Ueda, K., Okamoto, Y., & Yamawaki, S. (2016). Prediction of Immediate and Future Rewards Differentially Recruits Cortico-Basal Ganglia Loops. *Behavioral Economics of Preferences, Choices, and Happiness*, *7*(8), 593–616.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., Turck, F. D., & Abbeel, P. (2017). # Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 2753–2762). Vancouver, Canada.

Thrun, S. B. & Mitchell, T. M. (1993). Lifelong Robot Learning. In *NATO Advanced Study Institue on the Biology and Technology of Autonomous Intelligent Agents*. Trento, Italy.

van Hasselt, H., Guez, A., Hessel, M., Mnih, V., & Silver, D. (2016). Learning Values Across Many Orders of Magnitude. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 4287–4295). Barcelona, Spain.

van Seijen, H. & Sutton, R. S. (2014). True Online TD (λ). In *International Conference on Machine Learning (ICML)* (pp. 692–700). Beijing, China.

Vasan, G. & Pilarski, P. M. (2018). Context-Aware Learning from Demonstration : Using Camera Data to Support the Synergistic Control of a Multi-Joint Prosthetic Arm. In *IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)* (pp. 199–206). Enschede, The Netherlands.

Veeriah, V., Hessel, M., Xu, Z., Lewis, R., Rajendran, J., Oh, J., van Hasselt, H., Silver, D., & Singh, S. (2019). Discovery of Useful Questions as Auxiliary Tasks. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 9306–9317). Vancouver, Canada.

White, A. (2015). *Developing a Predictive Approach to Knowledge* (Doctoral dissertation, University of Alberta).

White, A., Modayil, J., & Sutton, R. S. (2014). Surprise and Curiosity for Big Data Robotics. In *AAAI Workshop on Sequential Decision Making with Big Data* (pp. 19–23). Quebec City, Canada.

White, M. (2017). Unifying Task Specification in Reinforcement Learning. In *International Conference on Machine Learning (ICML)*. Sydney, Australia.

White, M. & White, A. (2010). Interval Estimation for Reinforcement-Learning Algorithms in Continuous-State Domains. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 2433–2441). Vancouver, Canada.

White, M. & White, A. (2016). A Greedy Approach to Adapting the Trace Parameter for Temporal Difference Learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (pp. 557–565). Singapore.

Xu, Z., van Hasselt, H., & Silver, D. (2018). Meta-Gradient Reinforcement Learning. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 2396–2407). Montreal, Canada.

Yao, H., Szepesvári, C., Sutton, R., Modayil, J., & Bhatnagar, S. (2014). Universal Option Models. In *Conference on Neural Information Processing Systems (NeurIPS)* (pp. 990–998). Montreal, Canada.

Yu, H. (2015). On Convergence of Emphatic Temporal-Difference Learning. In *Conference on Learning Theory (COLT)* (pp. 1724–1751). Paris, France.

Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. *arXiv*, *1212.5701*.

Zhang, J., Springenberg, J. T., Boedecker, J., & Burgard, W. (2017). Deep Reinforcement Learning with Successor Features for Navigation across Similar Environments. In *The International Conference on Intelligent Robots and Systems (IROS)* (pp. 2371–2378). Vancouver, Canada.

Zhu, Y., Gordon, D., Kolve, E., Fox, D., Fei-Fei, L., Gupta, A., Mottaghi, R., & Farhadi, A. (2017). Visual Semantic Planning using Deep Successor Representations. In *IEEE International Conference on Computer Vision (ICCV)* (pp. 483–492). Venice, Italy.