Explorations in the Foundations of Value-based Reinforcement Learning

by

Kris De Asis

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

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

Value-based reinforcement learning is an approach to sequential decision making in which decisions are informed by learned, long-horizon predictions of future reward. This dissertation aims to understand issues that value-based methods face and develop algorithmic ideas to address these issues. It details three areas of contribution toward improving value-based methods. The first area of contribution extends temporal difference methods for fixed-horizon predictions. Regardless of problem setting, using fixed-horizon approximations of the return avoids the well-documented stability issues which plague off-policy temporal difference methods with function approximation. The second area of contribution introduces a framework of value-aware importance weights for off-policy learning and derives a minimum-variance instance of them. This alleviates variance concerns of importance sampling-based off-policy corrections. Lastly, the third area of contribution acknowledges a discrepancy between the discrete-time and continuous-time returns when viewing one as an approximation of the other, and proposes a modification to better align the objectives. This provides improved prediction targets, and when faced with variable time-discretization, improves control performance in terms of an underlying integral return.

Preface

Chapters 3, 4, and 5 of this dissertation are based on three published papers that I co-authored, all of which I am listed as the first author on.

Chapter 3 is based on De Asis et al. (2020), where I was responsible for the idea, stability arguments, empirical evaluation, and preparation of the paper. My co-authors contributed the proofs of convergence for linear and general function approximation, along with edits to strengthen the paper's presentation.

Chapters 4 and 5 are based on De Asis et al. (2023) and De Asis and Sutton (2024), respectively. For both papers, I was responsible for the idea, derivations, analysis, empirical evaluation, and preparation of the papers. My co-authors provided valuable feedback toward relevant literature and edits to improve each paper's presentation.

Acknowledgements

My graduate studies are among my most valued experiences. Contributing to this was the privilege to participate in and be surrounded by such a large and passionate research community. I would like to express by deepest gratitude to my supervisor Richard S. Sutton. Rich is a brilliant scientist and genuine truth-seeker who believes every idea should be appreciated for whatever it can offer. His attention to detail, constructive feedback, and unwavering support have been instrumental in my growth as a researcher.

I am extremely grateful to my supervisory committee, A. Rupam Mahmood and Harm van Seijen, and my examining committee, Michael Bowling, Adam White (candidacy), and Michael Littman, for providing me with technical feedback along with general research advice.

I owe much of my professional development to Daniel Graves for hosting me at Huawei and to Joseph Modayil for hosting me at Deepmind. Such opportunities to meet and work with a variety of different people produced invaluable discussion and novel perspectives on my work.

I extend my appreciation toward Alan Chan, Eric Graves, Yi Wan, Chen Ma, Arsalan Sharifnassab, Huizhen Yu, and the rest of the Reinforcement Learning & Artificial Intelligence laboratory at the University of Alberta. The group provides an inclusive and collaborative environment which encourages clear and careful work toward understanding the principles of intelligence.

I am lucky to have made and fostered many friendships beyond Academia: Nikita Quang, Karen Xu, Lindsey Peng, Yina Liu, Alex Tu, Devin Luu, Maitrayee Dhaka, Forte Shinko, Justin Jaffray, Francis Hermano, Russell Nicolas, Chris Pagal, Malcolm Ho, the local Albertan and broader Canadian speedcubing communities, the CELESTE Classic speedrunning community, the H-Mart

UA staff, and many more who have had an impact on me during my graduate studies. They never hesitate to extend support during tough times, allow me to pursue and share my hobbies, let me practice communicating ideas to diverse audiences, and provide fresh external perspectives that are relatively free of current biases and trends in the research community.

Last but definitely not least, I thank my sister Kathleen, my brother-in-law Louis, my parents Miguel and Hayley, my mother-in-law Chunhe, and my extended family in British Columbia. They provide continual unconditional support and nurture my growth as a person. Special thanks goes to my penguin Pengy, who since the beginning of my conscious life, somehow set me on a collision course with artificial intelligence. Extra special heartfelt thanks goes to my partner Youran Lin who in addition to understanding and always being there for me, is an absolutely inspiring academic.

Contents

List of Figures

Chapter 1

Introduction

The real world is vast and as a result it is generally impractical to account for every possible scenario. In applications like robotics, this vastness is often managed by engineering a controlled, artificial small world which prunes the space of situations one has to consider. Overcoming this need to design a small world with prohibitive operating conditions is important for the pursuit of creating generally intelligent systems. Acknowledging that one cannot account for everything, the world will appear *non-stationary* which necessitates the ability to learn from one's own stream of sensorimotor experience, and adapt to novel situations.

Reinforcement learning is a framework which formalizes an agent-environment interaction interface where an agent is to learn to predict and control from evaluative feedback. An agent generates a stream of interaction experience by continuously observing its situation, taking an action, and observing the outcome of this action. Among the observed outcome is a reward signal which provides a moment-by-moment evaluation of an agent's decisions and ultimately specifies an agent's objective of behaving to maximize the long-term accumulation of reward over some time horizon.

Value-based methods are an approach to reinforcement learning where predicted long-term consequences of an agent's actions are used to inform its decision-making. Specifically, a value-based agent learns to predict the expected future reward to result from each action, and given accurate predictions, can select the most promising action. A core idea underlying many value-based methods for reinforcement learning is *temporal difference* learning. Characteristic of temporal difference methods is the adjustment of one's predictions based on both observed immediate outcomes and predictions of what will follow beyond that. Trusting what is predicted to follow, an agent can immediately learn from every decision without waiting for long-term, eventual outcomes to occur.

Predictions about future reward by value-based agents can be interpreted as answering a behavior-conditional predictive question: Given that one behaves a certain way, how much reward can one expect to receive from this point onward, over some time horizon of interest? The *general value function* (GVF) framework extends these semantics for use with any signal of interest. Under this framework, a value-based agent can incrementally learn to predict the behavior-contingent long-term accumulation of any signal of interest, grounded in feedback from its stream of sensorimotor experience. Notably, all of such answers to predictive questions can be learned in parallel.

Value-based methods have seen great empirical successes over the years, from both directly informing behavior (Tesauro, 1995; Mnih et al., 2015) or by informing other components of a relatively complex system (Silver et al., 2016; Wurman et al., 2022). Despite this, when updating predictions with other predictions (*bootstrapping*), trying to predict what would have happened had one behaved differently (*off-policy learning*), and using *function approximation*, value-based methods may diverge. Instability in the presence of these three components is referred to as the *deadly triad*. It is well-documented with linear function approximation (Baird, 1995; Scherrer, 2010; Mahmood, 2017; Sutton and Barto, 2018) and has some empirical investigation with non-linear function approximation (van Hasselt et al., 2018). Several algorithms have been proposed to address the deadly triad (Sutton et al., 2009; Maei, 2011; Baird, 1995; Sutton et al., 2016; Mahmood, 2017), but practical considerations have been acknowledged surrounding high variance and relatively slow convergence (Baird, 1995; Hackman, 2012; Ghiassian et al., 2018; Ni, 2020).

In this dissertation, we aim to understand value-based methods in terms of practical concerns they may face as well as their promise in sequential decision making from evaluative feedback. Armed with this understanding, we further aim to develop algorithmic ideas to improve on or complement existing value-based techniques. We claim the following contributions:

1. Our first group of contributions proposes the use of **fixed-horizon approximations of the return** for stable reinforcement learning, regardless of problem setting. In particular:

    - We show that when using compositional GVFs with non-recursive bootstrapping, predicting up to a fixed-horizon avoids the deadly triad.
    - We characterize compositional GVFs in terms of their flexible predictive capabilities, and show how n-step bootstrapping, temporal abstraction, and parallelization can address computational concerns.
    - We empirically evaluate fixed-horizon methods to show convergence in a classic counterexample, understand the implications of a hard cut-off in a prediction horizon, and show improvement over an infinite-horizon algorithm in a control problem.
    - We propose a way in which behavior can explicitly take into account a large collection of fixed-horizon GVFs. We provide an implicit iterative-deepening intuition for it and compare it with only considering the furthest horizon.

2. Our second area of contribution surrounds using **value-based importance weighting** for off-policy corrections. Specifically:

    - We introduce the framework of value-based importance weighting, a broader class of multiplicative importance weights which consider a random variable's sample-space.
    - We derive a closed-form expression for a minimum-variance instance of these weights and we empirically characterize the variance reduction one can expect.
    - We derive a multi-step per-decision estimator to extend these weights for correcting sequences and use this estimator to extend existing off-policy algorithms.
    - We empirically evaluate off-policy algorithms with value-aware importance weights to demonstrate improved value estimation over conventional importance weighting.

3. Our third set of contributions identifies an **idiosyncratic dependence on time-discretization** which arises when naively applying discrete-time algorithms to discretized continuous-time environments. In this work:

- We view the discrete-time reinforcement learning objective as an approximation of its continuous-time analog and note that it computes an atypical Riemann sum.
- We argue that this atypical Riemann sum is a poor integral approximation in comparison with a conventional right-point Riemann sum and support the argument empirically.
- We propose a simple modification to the discrete-time objective to make it a right-point Riemann sum, resulting in improved prediction targets in terms of integral approximation error without loss of generality.
- We demonstrate the modification's improved control performance in the presence of variable time-discretization, in terms of maximizing the continuous-time objective.

The dissertation is organized as follows. Chapter 2 aims to provide relevant background information underlying the contributions of this thesis. Chapters 3, 4, and 5 correspond with and detail each of the three areas of contribution outlined above. Notably, the areas of contribution are independent from each other and as a result their corresponding chapters can be read in any order. Chapter 6 closes by providing concluding remarks in summary of the dissertation.

Chapter 2

Background and Problem Setup

This chapter provides background information, definitions, and notation to lay a foundation upon which the contributions of this dissertation will be communicated. It first introduces the reinforcement learning framework and the value-based approach to reinforcement learning. It then elaborates issues which plague off-policy temporal difference methods with function approximation and summarizes key algorithms which are motivated by addressing these issues. Readers familiar with these topics may choose to skip this chapter.

2.1 Reinforcement Learning

Reinforcement learning is a framework for learning from *evaluative feedback*. Evaluative feedback concerns evaluating *how good* an action or resulting situation is, in contrast with explicit instruction on what action(s) should have been taken. Central to reinforcement learning is the *agent-environment interface*. An agent exists in some state of the environment and receives some observation of this state. This observation can range from an identifier unique to every situation an agent may encounter to a vector containing features which summarize a situation. Given an observation, an agent takes an action in the environment which results in the environment transitioning to a new state and the agent receiving an observation of this new state along with a reward. These rewards define the objective of a reinforcement learning problem, where actions are to be selected in pursuit of maximizing the expected long-term sum of future rewards.

The agent-environment interface is typically formalized as a *Markov Decision Process* (MDP), where an agent interacts with the environment over discrete time steps. At each time step t, an agent observes the current state of the environment, $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all states in the MDP. An agent then uses this information to select an action $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(s)$ is the set of available actions in state s. The environment then jointly samples a next state, $S_{t+1} \in \mathcal{S}$, and a reward, $R_{t+1} \in \mathbb{R}$, according to the environment *transition model*:

$$p(s', r|s, a) \overset{\text{def}}{=} \Pr[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a].$$

Actions are selected according to a *policy*, a probability distribution over actions conditioned on state: $\pi(a|s) \overset{\text{def}}{=} \Pr[A_t = a|S_t = s]$. Following some policy, an agent is concerned with *returns*—discounted sums of future rewards:

$$G_t \overset{\text{def}}{=} \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... \tag{2.1}$$

given a discount factor $\gamma \in [0, 1]$ and T being the final time step in an episodic problem, or $\gamma \in [0, 1)$ and $T = \infty$ in a continuing problem. In reinforcement learning *control*, an agent is tasked with finding (and following) an optimal policy π^* which maximizes the *expected return* from each state.

2.2 Value-based Methods

Value-based methods for reinforcement learning aim to predict what the expected return will be and uses this information to inform decision making. More formally, they compute or estimate *value functions*, defined as the expected return when starting from a particular state (or state-action pair), and following policy π thereafter:

$$v_\pi(s) \overset{\text{def}}{=} \mathbb{E}_\pi[G_t | S_t = s]$$
$$q_\pi(s,a) \overset{\text{def}}{=} \mathbb{E}_\pi[G_t | S_t = s, A_t = a],$$

with v_π denoted the *state-value function* and q_π the *action-value function.* The problem of estimating a value function as accurately as possible is referred to as *policy evaluation.* Given a policy's action-value function, the *policy improvement* theorem (Sutton and Barto, 2018) states that deriving a new policy that is greedy with respect to these values will result in an improvement over the previous policy. To be precise, π_1 is an improvement over π_2 when $v_{\pi_1}(s) \geq v_{\pi_2}(s), \forall s \in \mathcal{S}$. Taken together, one can interleave policy evaluation and policy improvement in the process of *policy iteration* until the value function stops changing, settling at a unique *optimal value function* (denoted v^* or q^*) and a corresponding greedy optimal policy π^*.

One way to perform policy evaluation is to behave according to a policy and record the resulting returns from every visited state. This information can then be averaged into each state (or state-action pair)'s expected return. This describes a *Monte Carlo* method for policy evaluation, an example of a value-based method which learns solely from interaction experience in an environment. It is, however, limited by the need to observe complete (episodic) returns.

Another approach to policy evaluation makes use of a value function's *Bellman equation.* Such equations capture the relationship between a value at one decision point to the values of successor decision points. For example, $v_\pi(s)$ can be expressed as follows:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big(r + \gamma v_\pi(s')\big). \tag{2.2}$$

This gives a recursive, self-satisfying relationship where v_π is the unique fixed-point which satisfies it $\forall s \in \mathcal{S}$. Should one perform successive sweeps over the state space repeatedly applying the above operator with arbitrarily initialized value estimates, the estimates will approach the fixed-point:

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big(r + \gamma v_k(s')\big), \forall s \in \mathcal{S}. \tag{2.3}$$

This fixed-point iteration procedure gives the *dynamic programming* (Bellman, 1954; Howard, 1960) approach to policy evaluation, characterized by a value estimate *bootstrapping* off of the current value estimates of successor states. In contrast with the previously described Monte Carlo method, dynamic programming requires access to the environment transition model, $p(s', r|s, a)$.

The optimal value functions similarly have *Bellman optimality equations.* For q^*, we get:

$$q^*(s,a) = \sum_{s',r} p(s',r|s,a)\big(r + \gamma \max_{a'} q^*(s',a')\big).$$

Computing optimal value functions by means of dynamic programming with Bellman optimality equations gives the *value iteration* algorithm. Value iteration can be viewed as interleaving steps of policy evaluation and policy improvement at a fine granularity, where a greedy policy is derived regardless of whether policy evaluation has converged. The idea of performing policy improvement at any point is referred to as *generalized policy iteration* (Sutton and Barto, 2018) and provides a foundation for many value-based learning algorithms for control.

2.3 Temporal Difference Learning

Temporal difference (TD) methods (Sutton, 1988) approach policy evaluation by combining ideas from both Monte Carlo methods and dynamic programming. TD methods learn solely from streams of experience, akin to Monte Carlo methods, and TD methods bootstrap off of successor states' value estimates, like dynamic programming. The use of bootstrapping allows learning to occur without needing to observe complete returns, enabling *online*, incremental estimation of value functions. Given an approximate value function $V \approx v_\pi$ and an observed environment transition

$\{S_t, A_t, R_{t+1}, S_{t+1}\}$, one-step TD would compute the following estimate of the return:

$$\hat{G}_{t:t+1} \overset{\text{def}}{=} R_{t+1} + \gamma V(S_{t+1}). \tag{2.4}$$

This can be interpreted as combining a sampled immediate reward with a *guess* of what will happen from the next state onward to form an estimate for what happens from the previous state onward. Given this *TD target*, the previous state's value is updated with a step proportional to the difference between the target and its current estimated value:

$$V(S_t) \leftarrow V(S_t) + \alpha\big(\hat{G}_{t:t+1} - V(S_t)\big), \tag{2.5}$$

for some step size $\alpha \in [0, 1]$. Of note, the term in parentheses is referred to as the *TD error.* The target in Equation 2.4 can be viewed as a sample-based version of the dynamic programming update for v_π. Similarly, one can form a sample-based version of value iteration. For example, with q^*:

$$\begin{aligned} \hat{G}_t^{QL} &\overset{\text{def}}{=} R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\big(\hat{G}_t^{QL} - Q(S_t, A_t)\big). \end{aligned} \tag{2.6}$$

The resulting algorithm given by Equation 2.6 is the *Q-learning* algorithm (Watkins, 1989), arguably the most popular TD algorithm. It is notably an *off-policy algorithm*, because regardless of the policy driving behavior, the values will tend toward the optimal value function. The distinction between on- and off-policy will be further elaborated in Section 2.4. Based on generalized policy iteration, a Q-learning agent improves its policy by behaving greedily with respect to its estimates, but with a form of *exploration* to ensure every state-action pair is seen sufficiently often; e.g., an ϵ-greedy policy selects a uniform random action with probability ϵ, and the greedy action otherwise.

The target in Equation 2.4 being attributed to *one-step* TD alludes to the existence of *multi-step TD* methods. Here, one-step refers to the use of sampled information across one time-step, prior to bootstrapping to complete the estimate of the return. An n-step method instead stores information from the last n steps, discounts rewards appropriately, and then bootstraps:

$$\hat{G}_{t:t+n} \overset{\text{def}}{=} R_{t+1} + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... + \gamma^n V(S_{t+n}). \tag{2.7}$$

This target gives the n-step TD algorithm. Multi-step TD methods address a bias-variance trade-off (Jaakkola et al., 1994). Value estimates are largely inaccurate in the early stages of learning that bootstrapping off of them hinders learning progress relative to updates based on samples of what actually happened. Longer sequences of rewards, however, tend to have higher variance as the space of possible reward sequences can grow rapidly with sequence length.

n-step TD methods requiring information from n steps into the future introduces a delay in when learning can occur. To address this, and allow for smooth interpolation between One-step TD and the complete Monte Carlo return, one can consider the λ-return:

$$\hat{G}_t^{\lambda} \overset{\text{def}}{=} R_{t+1} + \gamma\big(\lambda \hat{G}_{t+1}^{\lambda} + (1-\lambda)V(S_{t+1})\big). \tag{2.8}$$

This return can be can be viewed as the expectation of a stochastic process where with probability λ, the return continues to the next sampled reward, and with probability $1-\lambda$, the reward sequence ends and the return estimate is completed via bootstrapping. Resolving the above recursion gives:

$$\hat{G}_t^{\lambda} = V(S_t) + \sum_{k=t}^{\infty} \big(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\big) \prod_{i=t+1}^{k} \gamma\lambda. \tag{2.9}$$

This suggests that the λ-return can be estimated by a weighted sum of one-step TD errors with later TD errors decayed at a rate of $\gamma\lambda$. The product notation in Equation 2.9 is used in place of exponentiation in anticipation of algorithms with transition-varying decay rates. Implementing this online and incrementally, one can maintain an *eligibility trace* summarizing the history of visited states with the weight of older states decaying at the aforementioned rate of $\gamma\lambda$. An agent can then attribute portions of the current one-step TD error to previous states through this eligibility trace and incrementally accumulate updates toward the λ-return. This procedure describes the TD(λ) family of algorithms (Sutton, 1988) which provide a computationally efficient approach to multi-step TD learning (van Hasselt and Sutton, 2015). It is worth noting that because values change on every time step, TD(λ) only approximates the λ-return. With linear function approximation, equivalence can be maintained via *true online TD(λ)* algorithms (van Seijen et al., 2016).

2.4 Off-policy Policy Evaluation

When tasked with policy evaluation for a policy that is *different* from the policy that was used to select actions, we face the *off-policy policy evaluation* problem. In this problem setting, the policy being evaluated is called the *target policy*, denoted π. The *behavior policy*, which drives behavior and generates data, is denoted μ. The *on-policy* setting is the special case where $\mu = \pi$.

A key idea underpinning many off-policy algorithms is the use of *importance sampling* (Hammersley and Handscomb, 1964; Rubinstein, 1981), an importance weight defined to be the ratio of the probabilities of an outcome under the sampling (behavior) and target distributions:

$$\rho_t \overset{\text{def}}{=} \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}. \tag{2.10}$$

Under an assumption of *coverage*, where $\mu(a|s) > 0$ whenever $\pi(a|s) > 0$, we have that:

$$\mathbb{E}_\mu[\rho_t|S_t = s] = \sum_a \mu(a|S_t)\frac{\pi(a|S_t)}{\mu(a|S_t)} = \sum_a \pi(a|S_t) = 1.$$

Given some action-dependent random variable X, we further have that:

$$\mathbb{E}_\mu[\rho_t X_t|S_t = s] = \sum_a \mu(a|S_t)\frac{\pi(a|S_t)}{\mu(a|S_t)}x = \sum_a \pi(a|S_t)x = \mathbb{E}_\pi[X_t|S_t = s].$$

This emphasizes that one can re-weight samples drawn from μ such that they average into the expected value under π. In correcting sequences of actions, as typical in reinforcement learning, one can use products of importance sampling ratios. This is evident from comparing the probability of a sequence of actions occurring under each policy:

$$\prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\mu(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{\mu(A_k|S_k)} = \prod_{k=t}^{T-1} \rho_k.$$

Given this, a Monte Carlo algorithm for off-policy policy evaluation could perform the following update with returns observed while following μ in order to estimate v_π:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big(\prod_{k=t}^{T-1} \rho_k\Big)\big(G_t - V(S_t)\big).$$

We note that the above update re-weights the *errors* as opposed to the return targets. Such a placement works as the subtracted value is independent from the corrected actions. The impact of this choice is further explored in Graves and Ghiassian (2022). While importance sampling can enable off-policy learning, it is well known to be of very high variance (Liu et al., 2020), leading to stability issues in practice and various proposals for addressing it (Rubinstein, 1981; Munos et al., 2016; Mahmood, 2017; Espeholt et al., 2018; Nachum et al., 2019; Zhang et al., 2020).

2.5 Value-based Methods with Function Approximation

For conceptual clarity, the ideas presented thus far have been for a tabular setting where each state has a unique identifier and values can be perfectly represented. However, the state space is often impractically large with agents only observing a snapshot of the underlying state through a set of features. This necessitates function approximation to more compactly represent a value function.

A value function is parametrized by a set of weights $\mathbf{w}$, $\hat{v}(S_t; \mathbf{w}_t)$, and the state is given as a set of features $\mathbf{x}(S_t)$ which may be raw observation signals or computed by some other feature construction process. An objective is then defined to establish a notion of a best set of weights. For value function approximation, a common choice is the *mean-squared value error* (MSVE):

$$\mathrm{MSVE}(\mathbf{w}) \overset{\text{def}}{=} \frac{1}{2} \sum_{s \in \mathcal{S}} d_\mu(s)\big(v_\pi(s) - \hat{v}(s; \mathbf{w})\big)^2, \tag{2.11}$$

where $d_\mu(s)$ gives the fraction of time one expects to spend in state s under behavior policy μ. Assuming that the value function parametrization is differentiable with respect to its weights, one can minimize the above objective with *stochastic gradient descent* (Widrow and Hoff, 1960). In stochastic gradient descent, a sample of the objective's gradient is computed (with respect to the weights) and the weights are adjusted by some amount along this direction:

$$\begin{aligned}\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} \mathrm{MSVE}(\mathbf{w}_t) \\ \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha\big(v_\pi(S_t) - \hat{v}(S_t; \mathbf{w}_t)\big)\nabla_{\mathbf{w}} \hat{v}(S_t; \mathbf{w}_t),\end{aligned} \tag{2.12}$$

where one can substitute a sampled estimate of the return in place of $v_\pi(S_t)$ if one does not have access to the true values. For example, with a one-step TD target:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha\big(R_{t+1} + \gamma \hat{v}(S_{t+1}; \mathbf{w}_t) - \hat{v}(S_t; \mathbf{w}_t)\big)\nabla_{\mathbf{w}} \hat{v}(S_t; \mathbf{w}_t). \tag{2.13}$$

It is important to note that the above update ignores the dependence of the bootstrapping term on the weights, making it a *semi-gradient* (Sutton and Barto, 2018) algorithm which performs approximate dynamic programming, as opposed to following the gradient of an objective.

2.6 The Deadly Triad and Off-policy Divergence

Off-policy TD methods with function approximation are plagued with what has come to be known as the *deadly triad* (Baird, 1995; Sutton and Barto, 2018): when coupling *off-policy learning* with *bootstrapping* and *function approximation*, learning updates may diverge.

Below is the TD update with linear function approximation, with feature vectors $\mathbf{x}$:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha\big(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t\big)\mathbf{x}_t. \tag{2.14}$$

Its expected update is of the form:

$$\mathbf{w}_{t+1} \leftarrow (\mathbf{I} - \alpha\mathbf{A})\mathbf{w}_t + \alpha\mathbf{b}, \tag{2.15}$$

where $\mathbf{A} = \mathbb{E}[\mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top]$ and $\mathbf{b} = \mathbb{E}[R_{t+1}\mathbf{x}_t]$, with expectations over both the transition dynamics and behavior policy's sampling distribution over states. Defining $\mathbf{X}$ to be a matrix where each row represents a state's feature vector, $\mathbf{D}$ to be a diagonal matrix with the probability of sampling each state on its diagonal, and $\mathbf{P}_\pi$ to be a matrix containing state-transition probabilities under the target policy, we have that $\mathbf{A} = \mathbf{X}^\top\mathbf{D}(\mathbf{I} - \gamma\mathbf{P}_\pi)\mathbf{X}$. In Equation 2.15, we can see that if $\mathbf{A}$ were a diagonal matrix with negative elements, the $\mathbf{I} - \alpha\mathbf{A}$ will amplify the corresponding weights and diverge. In contrast, if all elements were positive, the step size can be appropriately set to ensure the weight update continually shrinks and remains stable. More formally, the expected update is stable if $\mathbf{A}$ is positive definite. When on-policy, it is shown to always be positive definite (Sutton, 1988). This is not the case when off-policy due to potential mismatch between $\mathbf{D}$ and $\mathbf{P}_\pi$ (Baird, 1995; Tsitsiklis and Van Roy, 1997; Mahmood, 2017)— such mismatch leads to divergence.

One of the most well known examples of the deadly triad is *Baird's Counterexample* (Baird, 1995; Sutton and Barto, 2018), a small MDP with features for use with linear function approximation (Figure 2.1). The true values are completely representable and yet divergence occurs even when computing expected updates over the transition dynamics.

Several algorithms have been proposed to address these divergence issues. For example, the *residual-gradient* algorithm (Baird, 1995) performs gradient descent in the mean-squared TD error:

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma\hat{v}(S_{t+1}; \mathbf{w}_t) - \hat{v}(S_t; \mathbf{w}_t) \\ \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha\delta_t(\nabla_{\mathbf{w}}\hat{v}(S_t; \mathbf{w}_t) - \gamma\nabla_{\mathbf{w}}\hat{v}(S_{t+1}; \mathbf{w}_t)). \end{aligned} \tag{2.16}$$

The above update resembles that of semi-gradient TD in Equation 2.13 but accounts for the dependence of the bootstrapping term on the weights. Being a true gradient descent algorithm, it enjoys robust convergence guarantees, but has been acknowledged to be empirically slow and to

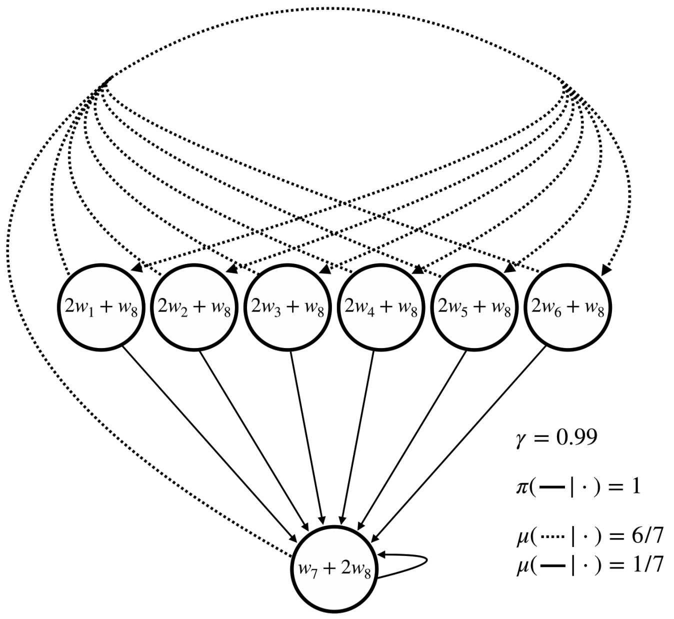


Figure 2.1: Baird's counterexample.

converge to arguably less-desirable solutions (Scherrer, 2010; Sutton and Barto, 2018).

Gradient TD (GTD) methods (Sutton et al., 2009; Maei, 2011) were later proposed. Like the residual gradient algorithm, they are true gradient descent algorithms and thus have convergence guarantees, but alternatively minimize the *mean-squared projected Bellman error*. This objective's gradient involves a product of expectations, two of which are not independent:

$$\nabla \text{MSPBE}(\mathbf{w}) = 2\mathbb{E}[\rho_t(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t)\mathbf{x}_t^\top]\mathbb{E}[\mathbf{x}_t\mathbf{x}_t^\top]^{-1}\mathbb{E}[\rho_t\delta_t\mathbf{x}_t] \tag{2.17}$$

Sutton et al. (2009) resolves the double sampling issue by estimating the expectation of the latter two factors at a slower time scale and combining it with a sample of the first. Based on algebraic manipulations of the first expectation, two (linear) GTD algorithms were proposed— *GTD2*:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha\rho_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})\mathbf{x}_t^\top \mathbf{v}_t, \tag{2.18}$$

and the *TD with gradient correction* (TDC) algorithm:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha\rho_t(\delta_t\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\mathbf{x}_t^\top\mathbf{v}_t), \tag{2.19}$$

where for both GTD algorithms, $\mathbf{v}_t$ is updated by:

$$\mathbf{v}_{t+1} \leftarrow \mathbf{v}_t + \beta\rho_t(\delta_t - \mathbf{v}_t^\top\mathbf{x}_t)\mathbf{x}_t.$$

GTD algorithms are the best understood of the stable off-policy TD methods, but similar to residual gradient methods, have been noted to converge slowly (Sutton and Barto, 2018; Sharifnassab and Sutton, 2023) in comparison with their semi-gradient counterparts. To address this, work has been done on hybrid methods which try to balance semi-gradient TD's convergence rate with GTD's stability (Hackman, 2012; White and White, 2016).

Another proposed solution to the deadly triad is the use of *Emphatic TD* (ETD) algorithms (Sutton et al., 2016; Mahmood, 2017). Because the deadly triad can be viewed as a mismatch between the sampling distribution over states $\mathbf{D}$ and the target policy's transition probabilities $\mathbf{P}_\pi$, ETD aims to re-weight updates to be under a different sampling distribution over states:

$$\begin{aligned} M_t &\leftarrow \gamma\rho_{t-1}M_{t-1} + I_t \\ \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha M_t\rho_t\delta_t\nabla_{\mathbf{w}}\hat{v}(S_t;\mathbf{w}_t), \end{aligned} \tag{2.20}$$

where I_t is the *interest* in state S_t, specifying the state *emphasis* M_t in the re-weighted TD update. While the state interest can be set to make ETD convergent on Baird's counterexample under expected updates, high variance in the sample-based update renders ETD inconsistent on the problem (Sutton and Barto, 2018).

2.7 The General Value Function Framework

Policy evaluation can be viewed as answering a predictive question about the future reward an agent expects to receive, conditioned on behavior. The *General Value Function* (GVF) framework (Sutton et al., 2011) generalizes value functions to represent answers to predictive questions about any signal of interest. Rewards are replaced by a more general *cumulant* C, which can be any quantity available to an agent, and the discount factor becomes a transition-dependent *continuation function* $\gamma_{t+1} \overset{\text{def}}{=} \gamma(S_t, A_t, S_{t+1})$ which specifies a prediction horizon. This gives:

$$v_\pi^{\gamma,c}(s) \overset{\text{def}}{=} \mathbb{E}_\pi\left[\sum_{k=0}^{T-t-1}\Big(\prod_{i=1}^{k}\gamma_{t+i}\Big)C_{t+k+1}\,\middle|\, S_t = s\right] \tag{2.21}$$

for state-value GVFs, but can similarly be conditioned on immediate action akin to action-values. A GVF has the semantics of answering the question: *How much of cumulant C do I expect to receive over some horizon into the future, given that I start in some state and follow policy π thereafter?* Consider a mobile robot with a GVF specified by a constant cumulant of 1 at every step, a continuation function that is 0 if a bump sensor is on (and 1 otherwise), and a target policy which always moves forward. Such a GVF would represent the expected number of time steps until the robot bumps into something should it keep moving forward from its current position. This example can be seen as a robot being able to learn a sense of *proximity* from a stream of sensorimotor experience which need not contain explicit distance information.

Perhaps the most promising aspect of GVFs is that a large collection of predictive questions can be answered in parallel from a single stream of experience (Sutton et al., 2011). Key to answering questions in parallel is the use of off-policy learning as it avoids the need to behave a specific way on a question-by-question basis. Function approximation is further needed due to the potentially vast number of questions on top of typically vast state spaces. Should an agent need to answer GVF questions online in a seemingly never-ending learning setup, one would need some form of bootstrapping. Such a context motivates addressing the deadly triad detailed in Section 2.6.

2.8 Temporal Difference Networks and Compositional Questions

The examples of GVF questions presented thus far have all been representable by a single value function due to having relationships which can be expressed recursively via state or transition-dependent quantities. *Temporal difference networks* (TD networks) (Sutton and Tanner, 2004; Tanner and Sutton, 2005) detail a framework for *compositional* questions. For example, one might ask about a cumulant over *exactly 2 steps*, as opposed to some transition-conditional horizon or an exponential window over the near future. This is accomplished by first predicting what happens in exactly one step (i.e., the cumulant's immediate value) and then predicting the sum of the cumulant's immediate value and what is predicted to happen one-step after that.

TD networks also entail *action-conditional* predictions, where a cumulant is masked based on whether a specific action was taken. This is comparable to action-value functions, but emphasizes compositions which answer questions involving sequences of actions, prior to following a policy.

TD networks, like standard TD, have temporal differences between the information that is composed in their TD targets. By explicitly maintaining several value functions which differ by temporal differences, they give an agent a notion of time. As a result, such compositional questions can produce an improved representation for non-Markovian problems (Sutton and Tanner, 2004). This is in line with the motivation for *predictive state representations* (Littman et al., 2001) and *general value function networks* (Schlegel et al., 2021) where collections of predictions about the future are used to cross representational gaps due to partial observability.

2.9 Finite Horizon MDPs

Much of what has been presented has been in the context of infinite-horizon and indefinite-horizon MDPs (continuing and episodic problems, respectively). There is a notable *finite-horizon MDP* setting (Puterman, 1994; Bertsekas and Tsitsiklis, 1996) where interaction with the environment occurs over a fixed number of time steps from a start state, e.g., a task where one is to gather as many resources as possible but return home within an hour. As such, an agent's optimal behavior in a state might change based on how much time has elapsed, prioritizing heading home as it nears the environment time limit. This time-varying behavior is referred to as a *non-stationary policy.* Of note, including elapsed time in the environment state gives an equivalent MDP with an optimal stationary policy. This emphasizes the finite-horizon setting's special treatment of time.

This problem setting has been well studied in the dynamic programming literature (Bertsekas, 2012), and from the value-based perspective, approaches the problem by predicting returns which account for the time remaining. Specifically, a value function for each fixed-horizon h up to the known environment time limit is computed, and are related by the following Bellman equation:

$$v_\pi^h(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big(r + \gamma v_\pi^{h-1}(s')\big), \tag{2.22}$$

where $v_\pi^0(s) \overset{\text{def}}{=} 0$ as nothing happens in 0 steps. This resembles the Bellman equation for v_π in Equation 2.2, but with the recursive relationship unrolled in time. Such a decomposition allows an agent in a finite-horizon problem to derive a greedy policy with respect to a value function with a horizon which matches the time remaining. While the work in this dissertation is *not* for this problem setting, a contribution of this dissertation shares similarities with this setting's solution method outlined above. The compositional relationship in Equation 2.22 can be well thought of as a TD network (See Section 2.8), with values notably being grounded in the 0th horizon.

Chapter 3

Fixed-Horizon Temporal Difference Methods for Stable Reinforcement Learning

In this chapter, we introduce the dissertation's first area of contribution: fixed-horizon temporal difference methods as a means to stabilize off-policy bootstrapping with function approximation. Specifically, we propose the use of fixed-horizon approximations of the return *regardless of problem setting*, and analyze how the compositional structure of the resulting TD network results in stable off-policy TD updates. We further elaborate many notable properties surrounding this space of algorithms including computational considerations, flexible predictive power, and algorithmic ideas which make explicit use of the additional information.

3.1 Fixed-horizon Returns and Values

A *fixed-horizon return* is a discounted sum of rewards similar to Equation 2.1, but only sums over a fixed number of future steps. For a horizon h, the fixed-horizon return is defined to be:

$$G_t^h = \sum_{k=0}^{\min(h,T-t)-1} \gamma^k R_{t+k+1}, \tag{3.1}$$

which is well-defined for any finite γ. This formulation allows more flexible characterization of an agent's horizon of interest and sense of urgency, and admits the use of $\gamma = 1$ in a continuing setting. Fixed-horizon value functions are then defined as expectations of the fixed-horizon return under policy π, starting from some state (or state-action pair):

$$v_\pi^h(s) = \mathbb{E}_\pi[G_t^h | S_t = s] \tag{3.2}$$

$$q_\pi^h(s,a) = \mathbb{E}_\pi[G_t^h | S_t = s, A_t = a]. \tag{3.3}$$

Like standard value functions, fixed-horizon value functions can also be expressed in terms of successor values. However, instead of bootstrapping off of the same value function, $v_\pi^h(s)$ bootstraps off of the successor state's value from a *shorter* horizon (Bertsekas, 2012):

$$v_\pi^h(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big(r + \gamma v_\pi^{h-1}(s')\Big),$$

where $v_\pi^0(s) = 0, \forall s \in \mathcal{S}$. While the resulting value function matches that of the finite-horizon MDP problem setting (Equation 2.22), we emphasize the distinction in proposing fixed-horizon values as a solution method for non-finite-horizon problems, i.e., h is not tied to a time limit. As an approximation to an infinite-horizon return, if an agent predicts a signal up to a final horizon H, it can flexibly specify a similar question with horizon $H + 1$ and directly make use of existing values. As $H \to \infty$ (and $\gamma < 1$), the fixed-horizon return converges to the infinite-horizon return, allowing fixed-horizon agents with sufficiently large H to solve infinite-horizon tasks arbitrarily well.

For a final horizon $H \ll \infty$, there may be concerns about suboptimal control. We explore this empirically in Section 3.6. We however note that optimality is never guaranteed when values are approximated. This can result from the use of function approximation, or even in tabular settings with a constant step size. Beyond this, van Seijen et al. (2019) has shown benefit in considering shorter horizons, based on mismatches in performance metrics. Li and Littman (2008) further provide strong guarantees for the finite-horizon setting while similarly suggesting the possibility of using finite-horizon solution methods as approximate solutions for other problem settings.

3.2 One-step Fixed-horizon TD

Similar to how TD methods can be seen as sample-based evaluation of Bellman equations, *fixed-horizon* TD (FHTD) methods are based on those of finite-horizon value functions (Equation 2.22). One-step FHTD would learn approximate values $V^h \approx v_\pi^h$ by computing, for each $h \in \{1, 2, ..., H\}$:

$$\begin{aligned} \hat{G}_t^h &= R_{t+1} + \gamma V^{h-1}(S_{t+1}) \\ V^h(S_t) &\leftarrow V^h(S_t) + \alpha\big(\hat{G}_t^h - V^h(S_t)\big), \end{aligned} \tag{3.4}$$

where again, $\forall s \in \mathcal{S}, V^0(s) \overset{\text{def}}{=} 0$. The idea of TD methods for predicting over a fixed interval dates back to Sutton (1988), and is acknowledged by work on TD networks (Sutton and Tanner, 2004). It was largely unexplored, seemingly due to increased computation for what appears to be a shorter-sighted objective. However, a key observation lies in how none of the value functions bootstrap recursively, and all eventually ground in $V^0(s) = 0$ which is fixed and *exactly known* from the start.

Considering control, we can similarly take a stochastic sample from the Bellman optimality equation for finite-horizon action-values. This gives fixed-horizon Q-learning (FHQ-learning):

$$\begin{aligned} \hat{G}_t^{QL,h} &= R_{t+1} + \gamma \max_{a'} Q^{h-1}(S_{t+1}, a') \\ Q^h(S_t, A_t) &\leftarrow Q^h(S_t, A_t) + \alpha\big(\hat{G}_t^{QL,h} - Q^h(S_t, A_t)\big), \end{aligned} \tag{3.5}$$

with $Q^0(s, a) \overset{\text{def}}{=} 0, \forall (s, a) \in \mathcal{S} \times \mathcal{A}$. The action-values represent the largest returns achievable within a horizon: The largest h-horizon return from a state-action pair is the immediate reward and the best one can do in $h - 1$ steps from the next state. As each horizon generally has a different maximizing action in a state, fixed-horizon control is *inherently off-policy.* When deriving behavior from fixed-horizon action-values, a reasonable choice is to be approximately greedy with respect to the furthest horizon being learned about, with a form of exploration to ensure state space coverage. We explore this, and an alternative which takes every horizon into account, in Section 3.6.

3.3 Stability of the FHTD Update

Such non-recursive bootstrapping allows FHTD methods to still "learn a guess from a guess," (Sutton and Barto, 2018) but assuming parameters are not shared between horizons, avoids the deadly triad when learning off-policy with function approximation. Intuitively, the first horizon is predicting the immediate reward via a Monte Carlo method, which is a stable target and should ultimately converge. Once this has converged, the second horizon, which bootstraps off of the first horizon, will eventually have a stable target, and so forth. If we write a horizon's expected linear update in the form of Equation 2.15, we get:

$$\mathbf{w}_{t+1}^{h} \leftarrow (\mathbf{I} - \alpha\mathbf{A})\mathbf{w}_{t}^{h} + \alpha\mathbf{b}^{h}, \tag{3.6}$$

where $\mathbf{A} = \mathbb{E}[\mathbf{x}_t\mathbf{x}_t^\top]$, $\mathbf{b}^h = \mathbb{E}[\big(R_{t+1} + \gamma\mathbf{w}_t^{h-1} \cdot \mathbf{x}_{t+1}\big)\mathbf{x}_t]$, and expectations are over the transition dynamics and sampling distribution over states. The matrix $\mathbf{A}$ is positive definite, and thus the update avoids divergence due to a mismatch between the state sampling distribution and the transition dynamics under a different target policy. We instead have a Monte Carlo-like update using a changing but ultimately converging vector $\mathbf{b}^h$.

Beyond the above analysis, De Asis et al. (2020) proves convergence for FHQ-learning with linear function approximation by showing that the update's corresponding ODE converges with probability 1 to a globally asymptotically stable equilibrium. The equilibrium's existence assumes that earlier horizons converge quicker than later ones, which is expected given the direction of bootstrapping dependencies. This is less limiting than the corresponding assumption in Melo and Ribeiro (2007)'s proof for linear Q-learning, which requires the behavior policy to be near-optimal.

3.4 Flexibility of the Composition

Given the relatively large set of value functions learned, an FHTD agent has more predictive capabilities than that of a TD agent looking at a comparable horizon specified via discounting. Such fixed-horizon returns can be viewed as an inner product between the reward signal and a basis of step functions, giving an agent an exact notion of when rewards occur, as it can subtract values of subsequent horizons to extract each step's expected reward.

Further, the unidirectional dependencies suggest that if an agent has already learned reasonably accurate values up to a horizon of H, and needs to look farther into the future, it can specify a horizon of $H+1$ and bootstrap off of what has already been learned. This ability to re-use values of previously learned horizons similarly extends to control, as the maximizing action for a horizon only depends on the preceding horizons. If an agent periodically trusts that its H-horizon predictions have converged, and "freezes" this value function for a newly specified $H+1$-horizon to bootstrap off of, this resembles a fitted-value iteration (Riedmiller, 2005) or target network (Mnih et al., 2015) procedure and provides an *iterative-deepening of horizon* interpretation of them.

Having the hard cut-off of a fixed-horizon and knowing what each value function's horizon is opens up significant flexibility in temporal weighting of the return. For example, one may use this information to directly learn the *fixed-horizon average reward* with the following TD target:

$$\frac{1}{h}R_{t+1} + \frac{h-1}{h}V^{h-1}(S_{t+1}). \tag{3.7}$$

This may have implications for non-linear function approximation, should values of a similar magnitude be easier to represent (van Hasselt et al., 2016). Further exploring the flexibility of re-weighting the immediate reward, we can get the following alternative TD target for exponential discounting:

$$\gamma^{H-h}R_{t+1} + V^{h-1}(S_{t+1}). \tag{3.8}$$

It can be observed that for $\gamma < 1$, earlier horizons will have values close to zero, putting more reliance on later horizons being correct. This contrasts the exponentially-discounted FHTD target in Equation 3.4 which puts larger reliance on earlier horizons being correct. Further exploring reward scaling, one can directly learn time-inconsistent preferences like hyperbolic-discounting (Sozou, 1998; Fedus et al., 2019) which notably *requires* a hard cut-off as the harmonic series diverges:

$$\frac{1}{1+k(H-h)}R_{t+1} + V^{h-1}(S_{t+1}). \tag{3.9}$$

3.5 Computational Considerations

A drawback of the FHTD methods presented thus far revolves around computation scaling linearly with horizon. That is, an agent has to store and regularly update H value functions. If we consider a case where we have no explicit use for the earlier horizons, and only need to predict what happens in H steps, one can use a Monte Carlo method which stores and sums the last H rewards to directly learn a *single* value function. This suggests that n-step FHTD methods bridge a computational trade-off (in addition to one of bias-variance). With storage of the last n rewards, one can directly estimate V^n. With the option to bootstrap off of V^n, one can form a target for V^{2n}, and so forth. From this, an n-step FHTD method only needs to learn and update $\lceil \frac{H}{n} \rceil$ value functions.

Let us consider a tabular setting where querying a state's approximate value is cheap and assume memory is abundant. Focusing on the operations of adding up (and appropriately discounting) rewards and updating an entry in a value function, both one-step FHTD and the fixed-horizon Monte Carlo algorithm perform H such operations. n-step FHTD performs $n + H/n$ operations, which is less than H (for $1 < n < H$) and has a best case of $2\sqrt{H} - 1$ operations at $n = \sqrt{H}$. While other considerations exist (e.g., memory constraints, cost of querying a value function with function approximation), this suggests that sub-linear scaling with horizon is possible. See Appendix A for pseudocode describing n-step FHTD. We further acknowledge that any use of temporal abstraction, such as the options framework (Sutton et al., 1999; Precup, 2000) or "down-sampling" an MDP via frame-skipping (Bellemare et al., 2013; Mnih et al., 2015), can further substantially reduce the computation required to predict H time-steps into the future.

Akin to prior work with large collections of GVFs (Sutton et al., 2011), such fixed-horizon predictions can be learned in *parallel* from a stream of experience. In the context of neural network function approximation, one might represent horizons as outputs over a shared representation. Such an architecture's computation can be accelerated by a GPU with readily available frameworks (Abadi et al., 2016; Bradbury et al., 2018; Paszke et al., 2019), and memory-wise, this increase in parameters tends to be minimal compared to the rest of the network. Unfortunately, such a shared representation violates the non-recursive bootstrapping required for stability. If one views the aforementioned architecture as representation learning layers with a linear function over the resulting representation, the value functions are separate from the perspective of the linear output layer. Should the representation settle or vary slowly (e.g., layer-based step sizes to operate at different time-scales), the stability arguments apply.

3.6 Empirical Evaluation

To evaluate FHTD methods, we ran the following experiments:

- Off-policy prediction with linear function approximation in Baird's Counterexample to validate and emphasize stability in the presence of the deadly triad.
- A comparison between (tabular) discounted Q-learning and undiscounted FHQ-learning in a stochastic grid world, to understand differences between hard and soft-termination of the return over similar effective horizons.
- A comparison between DQN and fixed-horizon DQN (DFHQ) in the *LunarLander-v2* environment (Brockman et al., 2016), to demonstrate the practicality of FHTD in a deep RL setting, and to see if intuitions hold when weights are shared through a learned representation.
- A comparison between different ways to derive behavior in the presence of a large collection of fixed-horizon values to see if one can capitalize on the additional information available.

3.6.1 Stability in Baird's Counterexample

In Baird's Counterexample (Figure 2.1), we ran one-step FHTD with importance sampling corrections and tried to predict up to $H = \frac{1}{1-\gamma} = 100$. Every horizon's weight vector was initialized to $\mathbf{w}^h = [1, 1, 1, 1, 1, 1, 10, 1]^\top$ based on Sutton and Barto (2018) and we used step size $\alpha = \frac{0.2}{|\mathcal{S}|}$. We logged the trajectory of the 100th horizon's weight vector over 1,000 independent runs of 10,000 steps, and plotted the results in Figure 3.1.

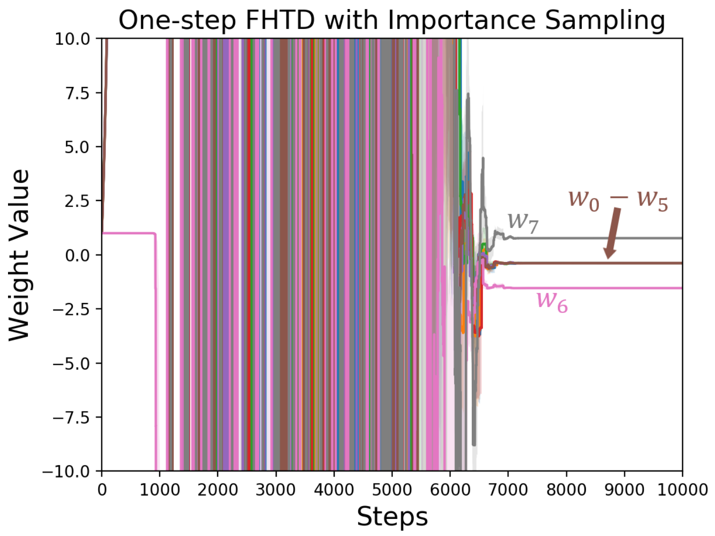


Figure 3.1: Weight trajectories of one-step FHTD's 100th horizon value function on Baird's counterexample, plotted after each time step. Results are averaged over 1,000 runs, and shaded regions represent one standard error.

We can see that despite substantial oscillations, FHTD consistently converges. Notably, the resulting weight vector is optimal, and consistent with existing comparisons between semi-gradient and gradient-based TD methods, FHTD converged relatively quickly (Sutton and Barto, 2018; Ghiassian et al., 2018). Regarding the oscillations, part of this can be attributed to every horizon being initialized to the same weight vector. Updating every horizon synchronously, the final horizon will resemble a recursive-bootstrapping TD method for at least $H - 1$ steps. In other words, the oscillations emphasize what semi-gradient TD would have done, followed by FHTD's ability to recover from this initial weight explosion.

Of note, the oscillations can be attenuated or completely avoided via asynchronous updates or iterative-deepening of horizons, as that can avoid the initial updates resembling those of semi-gradient TD. We emphasize this by comparing two fixed-horizon dynamic programming (FHDP) runs where expected updates are performed over 400 successive sweeps of the state space. The first performs synchronous updates akin to the previous experiment, while the latter reorders the updates to iteratively deepen the horizon— only one horizon is updated per sweep, and it moves to a subsequent horizon every 400 sweeps. Results can be found in Figure 3.2.

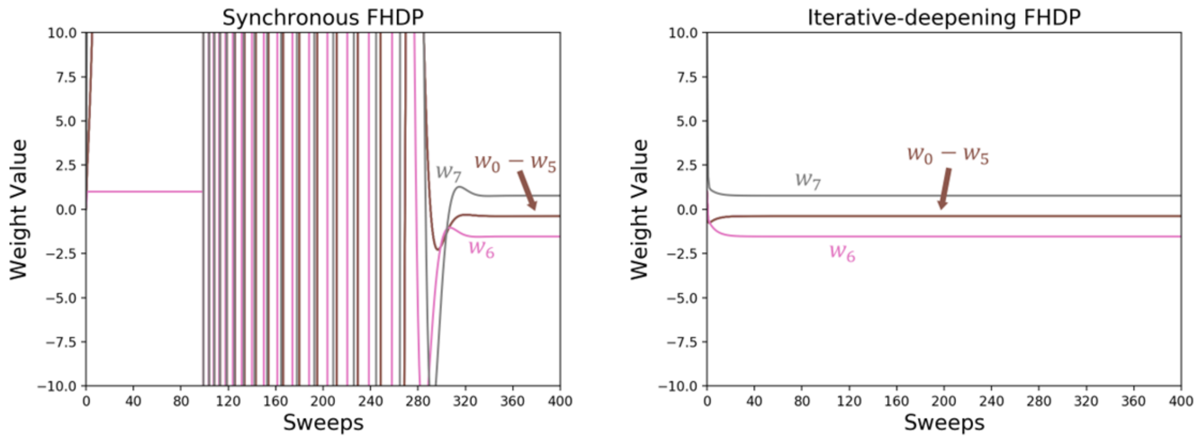


Figure 3.2: Weight trajectories of synchronous and iterative-deepening FHDP's 100th horizon value function on Baird's counterexample, across 400 sweeps of the state space.

It can be seen that synchronous FHDP exhibits similar oscillations to those of FHTD. However, by reordering the updates, one can completely eliminate the oscillations and likely get away with far fewer sweeps. While an online algorithm may be limited in how updates can be reordered, sequencing a given transition's updates by increasing horizon can make use of more up-to-date information should there be a self-looping transition or generalization between consecutive states. Using one-step FHTD with importance sampling under the same step size as the previous experiments, Figure 3.3 explores such update sequencing in Baird's counterexample by visualizing the RMS error in the 100th horizon's value function, averaged over 30 runs. We see that even with online updates, sequencing updates by horizon can dramatically attenuate the weight explosion.

3.6.2 Fixed vs. Discounted Horizons in Tabular TD control

Here we compare undiscounted FHQ-learning with discounted Q-learning in a *slippery maze*. This environment is a small gridworld with 4-directional movement, where moving into a wall keeps the agent in place. There's a fixed start and terminal state and a per-step reward of -1. The "slipperiness" entails a 75% chance that an agent's action is overridden by a uniform random

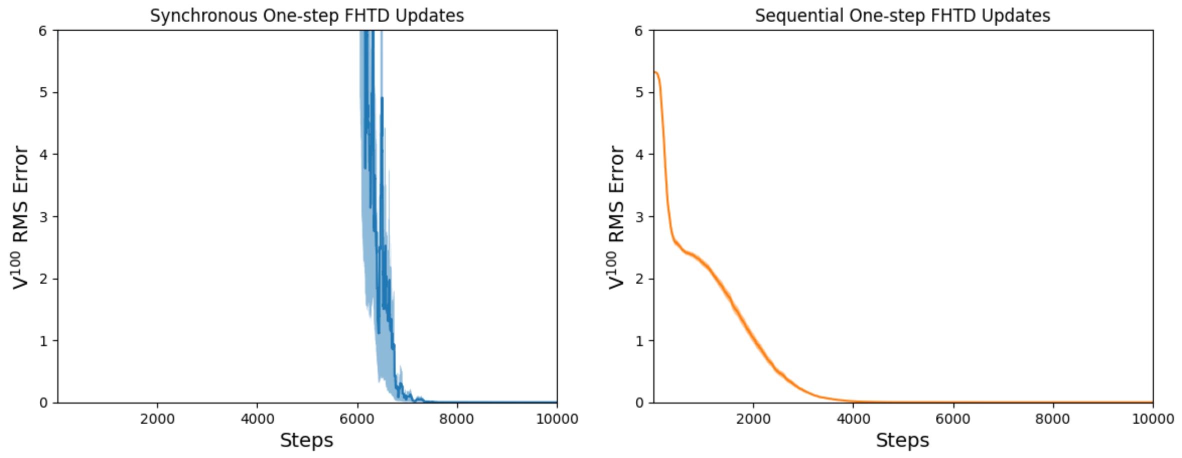


Figure 3.3: RMS error in FHTD's 100th horizon value function on Baird's counter example with synchronous and sequential updates per-state. Results are averaged over 30 runs, and shaded regions represent one standard error.

action unbeknownst to the agent. The optimal deterministic path is 14 steps, but due to the stochasticity, an optimal policy is expected to take ≈ 62 steps to reach the goal.

Such a highly stochastic environment emphasizes methods which consider shorter horizons, as the uncertainty in long term information might provide more variance than useful signal to incorporate in decision making. For FHQ-learning, we considered $H \in \{8, 16, 32, 48\}$, and selected discount factors $\gamma \in \{0.875, 0.938, 0.969, 0.979\}$ for Q-learning. These discount factors were chosen to match H in terms of expected sequence length of a stochastic process with termination probability $1 - \gamma$. Both agents behaved ϵ-greedily with $\epsilon = 0.1$, where FHQ-learning did so with respect to its furthest predicted horizon. It is hypothesized that despite the correspondence between the fixed-horizons and discount factors chosen, discounting would still include a significant amount of information beyond its expected horizon. This might perform better than a fixed-horizon when H is unreasonably short for the problem, but might perform worse than a sufficiently long H by including additional, highly-variable information beyond this point. We performed 100 independent runs for each parameter setting, and Figure 3.4 shows the mean episode length after 100 episodes.

For FHQ-learning, it is evident that if H is unreasonably small ($H = 8$), the agent struggles as it must first randomly stumble on a situation that is H-steps from the goal. However, $H = 16$ does considerably better than if it were to predict further into the future, be it with a longer

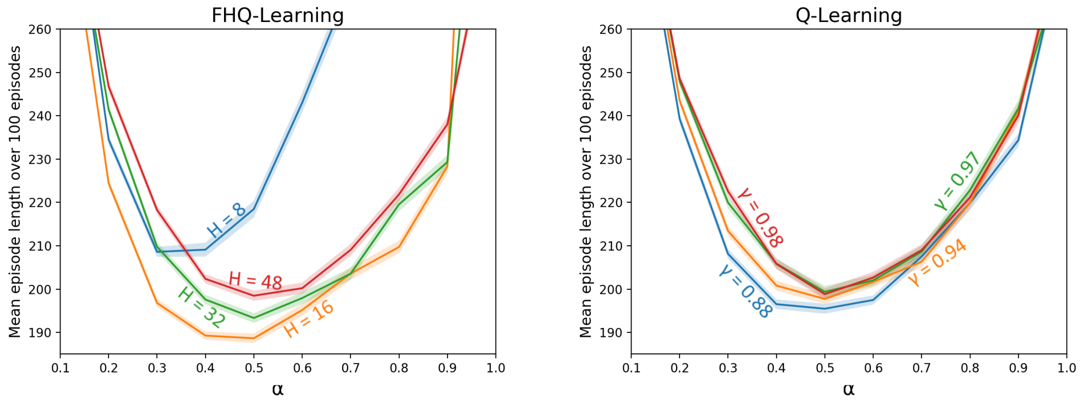


Figure 3.4: Mean episode lengths over 100 episodes of FHQ-learning and Q-learning with various step-sizes and horizons of interest. Results are averaged over 100 runs, and shaded regions represent one standard error.

fixed-horizon or with an expected horizon induced by discounting. With Q-learning, each discount factor performed relatively similar to one another, despite them being expected to correspond with the tested values of H. This is in support of discounting including a considerable amount of highly variable information beyond H steps. As expected from the properties of the environment, a shorter horizon (i.e., small γ or small H) was preferable over the full episodic return for both algorithms.

3.6.3 Deep FHTD Control

In this experiment, we tested a fixed-horizon version of DQN (Mnih et al., 2015) (DFHQ). We treated each horizon's values as linear outputs over shared hidden layers in a neural network. Use of this architecture makes it very amenable to parallelization on a GPU, and emphasizes how the increased computation can be managed. Using sampled mini-batches from an experience replay buffer, DFHQ minimizes the following objective:

$$J(\mathbf{w}) = \frac{1}{H}\sum_{h=1}^{H}\Big(\hat{G}_t^h - Q^h(S_t, A_t; \mathbf{w})\Big)^2, \tag{3.10}$$

where, ignoring the target's dependence on the weights via a stop-gradient operator,

$$\hat{G}_t^h = R_{t+1} + \gamma \max_{a'} Q^{h-1}(S_{t+1}, a'; \mathbf{w}).$$

We compared against DQN in the LunarLander-v2 environment (Brockman et al., 2016). Each algorithm behaved ϵ-greedily where ϵ annealed from 1.0 to 0.1 over 50,000 frames, and DFHQ predicted up to $H = 64$. We performed 30 independent runs of 500,000 frames ($\approx$ 1,000 episodes) for each algorithm, sweeping over hidden layer widths, optimizer parameters, and discount factors. See Appendix A for complete details. Figure 3.5 shows for each frame, the mean return of the last 10 episodes for each algorithm's best identified parameters in terms of area under the curve.

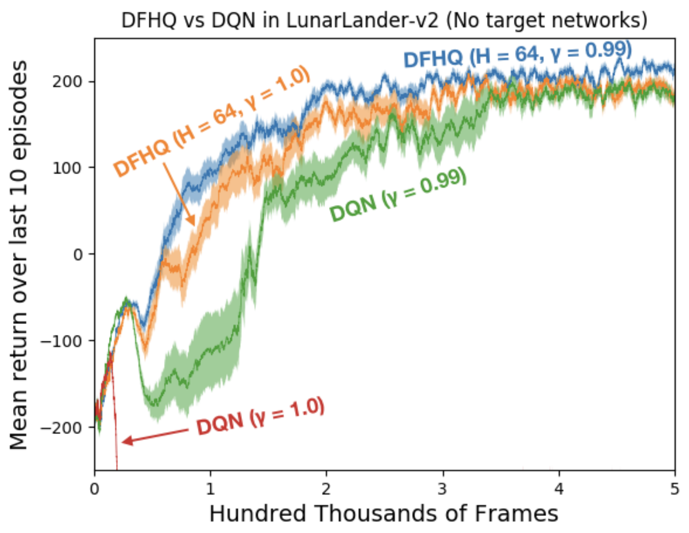


Figure 3.5: Mean return over last 10 episodes per frame for DFHQ and DQN, without target networks, averaged over 30 runs. Shaded regions represent one standard error.

Of note, the results show DFHQ and DQN *without* target networks. From additional experiments, we found that the delay induced by target networks slowed DFHQ's learning more than it could help over a run's duration, and hypothesize that the decoupling of bootstrapping targets in DFHQ alleviates the need for it. Target networks marginally improve DQN's performance, but the area under the curve remained well below that of DFHQ. It is evident in this environment that DFHQ had lower variance and relatively steady improvement. DQN with $\gamma = 1$ immediately diverged while DFHQ was less sensitive to γ, emphasizing the shift in γ's focus as a temporal preference (e.g., sense of urgency) than specifying the prediction horizon. From the remainder of our sweep, DFHQ also seemed insensitive to larger hidden layer widths beyond what is shown, whereas DQN's performance considerably degraded as the network size increased. These results support the computational practicality of the algorithm with non-linear function approximation, and that learning can still be stable with a shared representation despite the analysis requiring weights be separated by horizon. DFHQ's performance may be attributed to the representation learning benefits of predicting many auxiliary tasks with a shared representation (Jaderberg et al., 2017). In contrast with auxiliary tasks, such predictions are *necessary* tasks for the final horizon.

3.6.4 An Alternative Behavior Policy for FHTD Control

The previous FHTD control experiments used an ϵ-greedy policy with respect to the furthest horizon being learned about. This is reasonable as these fixed-horizons are used as an approximation to the infinite (or indefinite) horizon returns, and the furthest fixed-horizon is closest to it. Here, we consider a behavior policy where each horizon "votes" with their current greedy action, and the mode of these proposals is selected. Specifically, we define an ϵ-*democratic* policy which is akin to ϵ-greedy, but the greedy action is replaced by the aforementioned voting procedure.

The intuition for this procedure is based on earlier horizons converging more quickly than later horizons. As a result, they may come to a consensus more quickly than the later horizons and exploit near-term benefits. As learning progresses and later horizons converge, the further horizons will begin to agree. Should the final horizon be twice the required length for optimal behavior, the consensus of the further horizons will eventually dominate and produce an optimal policy. This can be interpreted as performing iterative-deepening of horizon in an agent's behavior, using consensus of the greedy action as a proxy for whether horizons have approximately converged. Furthermore, there may be additional stability from an ensemble-like approach.

We consider an undiscounted 5-state MDP where an agent starts on one end, and has two actions in each state: one action advances to the next state with reward -1, and the other immediately terminates with a terminal reward that increases further along the chain. Attempting to advance beyond the chain's length terminates with a large positive reward. This *HorizonChain* environment is structured such that shorter horizons will tend to favor early termination. A complete specification can be seen in Figure 3.6.

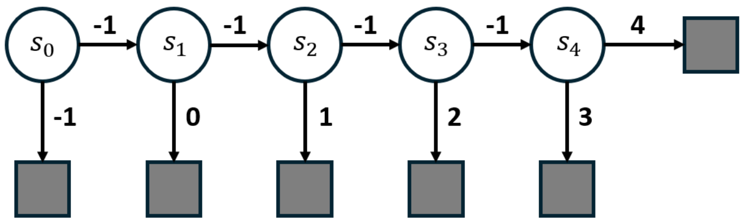


Figure 3.6: The 5-state HorizonChain MDP.

We compared ϵ-greedy and ϵ-democratic behavior policies with $\epsilon = 0.1$, using an FHQ-learning agent with step size $\alpha = 0.5$ and final horizon $H = 10$. In each episode's start state, we recorded the ϵ-democratic policy's *furthest* horizon among those who agreed with the majority greedy action. We hypothesize that this measure of *maximum effective horizon* should increase over time, based on the iterative-deepening intuition. We additionally measure the episodic return for each behavior policy. It is hypothesized that the ϵ-democratic policy should improve over ϵ-greedy from greedifying with respect to relatively more confident information. We performed 1,000 independent runs of 1,000 episodes and the results can be seen in Figure 3.7.

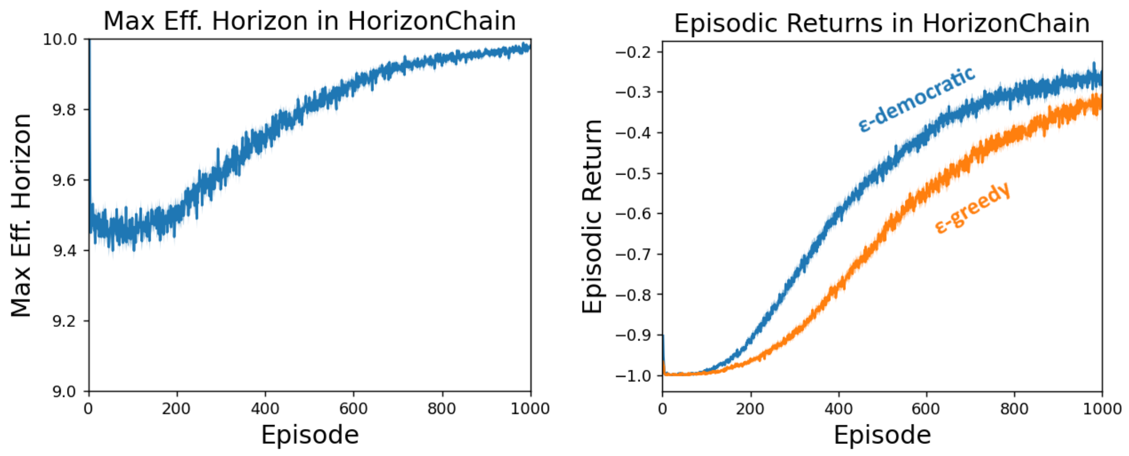


Figure 3.7: **Left**: Maximum effective horizon of the ϵ-democratic policy from each episode's start state. **Right**: Episodic returns for ϵ-democratic and ϵ-greedy behavior with FHQ-learning. Results are averaged over 1,000 runs, and shaded regions represent one standard error.

With regards to the maximum effective horizon over time, we see that it increases over time. It may seem minimal as it goes from a horizon of ≈ 9.5 toward 10, but we reiterate that this measure presents the *furthest* agreeing horizon. With only two actions, it is additionally very likely for later horizons to randomly agree despite having poor estimates, biasing the metric toward larger values. Nevertheless, the results are in line with the iterative-deepening intuition. We further see substantial improvement over ϵ-greedy in terms of episodic returns, emphasizing the promise of explicitly considering the additional information, be it for iterative deepening or ensemble benefits.

3.7 Summary

In this chapter, we investigated using *fixed-horizon* returns in place of the conventional infinite-horizon return. We derived FHTD methods and crucially argued that they are stable in the presence of the deadly triad. Such stability stems from non-recursive bootstrapping between a collection of compositional GVFs. We further emphasized the flexible predictive capabilities of this collection of GVFs and how it can directly learn arbitrary temporal preferences beyond what is possible with recursive bootstrapping. Empirically, we showed that off-policy linear FHTD converges on a classic counterexample for off-policy linear TD and emphasized how asynchronous updates and iterative-deepening of horizon can dramatically improve learning. In a tabular control problem, we compared undiscounted fixed-horizons with comparable effective horizons imposed by discounting, emphasizing the implications of a hard termination in the horizon of interest. We then demonstrated that FHTD methods can scale well to and perform competitively in a deep reinforcement learning control problem. Lastly, we presented a behavior policy which explicitly considers the additional information available (from the earlier horizons) when making decisions. By having horizons vote with their greedifying action, we argued that this procedure results in implicit iterative-deepening of horizon in behavior, provided empirical evidence in support of this intuition, and demonstrated improved control performance over only considering the furthest horizon.

Chapter 4

Value-aware Importance Weighting for Off-policy Reinforcement Learning

In this chapter, we detail the dissertation's second area of contribution: an alternative way to re-weight samples for off-policy learning. As detailed in Chapter 2, a common strategy for extending algorithms for use with off-policy data is the use of *importance sampling* (Kahn, 1950a,b; Kloek and Van Dijk, 1978). Importance sampling scales outcomes by the ratio of an action's probability under the target and behavior policies in order to produce an unbiased estimate of the outcome's expectation under the target policy. However, importance sampling-based off-policy estimators are well-known to suffer from high variance (Liu et al., 2020). We instead consider *value-aware importance weighting*. Like importance sampling, we correct samples by a multiplicative weight, but further take into account knowledge of the sample space to produce weights of lower variance.

We note that this work focuses on correcting the update target but not the state-visitation distribution, as it is a simpler setting where variance concerns already manifest. Nevertheless, we present preliminary results with Emphatic TD methods in Section 4.4.4 showcasing the generality of our estimator and its compatibility with methods which re-weight the state-visitation distribution. We further solely consider the case where action-values (and not state-values) are learned.

4.1 Value-aware Importance Weighting

Importance sampling (Equation 2.10) only depends on the behavior and target policy probabilities. This has appeal in being simple, interpretable, and generally applicable. In this section, we consider importance weights which additionally depend on a state's immediate action-values (denoted $\check{\rho}$). We view the space of importance weights as a constrained optimization problem subject to:

1. $\sum_a \mu_a \check{\rho}_a Q_a = \sum_a \pi_a Q_a$
2. $\sum_a \mu_a \check{\rho}_a = 1$

where for notational convenience, subscripts condition on a specific action and everything is implicitly conditioned on the same (arbitrary) state. The first constraint ensures that the weighted action-values sampled under the behavior policy provides a sample of the expected action-value under the target policy. The second constraint fixes the expected importance weight to be 1. This was specified as it is a property of importance sampling and avoids arbitrary attenuation or amplification in anticipation of needing to correct sequences. Given importance sampling's variance concerns, a reasonable objective is to minimize $\mathrm{Var}_\mu(\check{\rho})$. This leads to the following Lagrangian:

$$\mathcal{L}(\check{\rho}, \lambda_0, \lambda_1) = \sum_a \mu_a (\check{\rho}_a - 1)^2 + \lambda_0 \Big(\sum_a \pi_a Q_a - \sum_a \mu_a \check{\rho}_a Q_a \Big) + \lambda_1 \Big(1 - \sum_a \mu_a \check{\rho}_a \Big). \tag{4.1}$$

We can rewrite this in matrix form as follows:

$$\mathcal{L}(\check{\boldsymbol{\rho}}, \lambda_0, \lambda_1) = (\check{\boldsymbol{\rho}} - 1)^T D_\mu (\check{\boldsymbol{\rho}} - 1) + \lambda_0 \big(\boldsymbol{\pi}^T \mathbf{Q} - (D_\mu \mathbf{Q})^T \check{\boldsymbol{\rho}} \big) + \lambda_1 (1 - (D_\mu \mathbf{1})^T \check{\boldsymbol{\rho}}), \tag{4.2}$$

where $\boldsymbol{\mu}$, $\boldsymbol{\pi}$, and $\mathbf{Q}$ are column vectors containing, for each action in a given state, the entries of μ_a, π_a, and Q_a, respectively. D_μ is a diagonal matrix with $\boldsymbol{\mu}$ along its diagonal, and $\check{\boldsymbol{\rho}}$ is a state's vector of importance weights. We assume $\boldsymbol{\mu}$ has a non-zero probability for each action to ensure

D_μ's inverse exists. Setting $\nabla\mathcal{L}(\check{\boldsymbol{\rho}}, \lambda_0, \lambda_1) = 0$ and solving for $\check{\boldsymbol{\rho}}$, we get:

$$\begin{aligned}
\nabla\mathcal{L}(\check{\boldsymbol{\rho}}, \lambda_0, \lambda_1) = 0 &= 2D_\mu\check{\boldsymbol{\rho}} - 2\boldsymbol{\mu} - D_\mu\mathbf{Q}\lambda_0 - D_\mu\mathbf{1}\lambda_1 \\
0 &= D_\mu\check{\boldsymbol{\rho}} - \boldsymbol{\mu} - D_\mu\mathbf{Q}\frac{\lambda_0}{2} - D_\mu\mathbf{1}\frac{\lambda_1}{2} \\
0 &= D_\mu^{-1}D_\mu\check{\boldsymbol{\rho}} - D_\mu^{-1}\boldsymbol{\mu} - D_\mu^{-1}D_\mu\mathbf{Q}\frac{\lambda_0}{2} - D_\mu^{-1}D_\mu\mathbf{1}\frac{\lambda_1}{2} \\
0 &= \check{\boldsymbol{\rho}} - \mathbf{1} - \mathbf{Q}\frac{\lambda_0}{2} - \mathbf{1}\frac{\lambda_1}{2} \\
\check{\boldsymbol{\rho}} &= \mathbf{1} + \mathbf{Q}\frac{\lambda_0}{2} + \mathbf{1}\frac{\lambda_1}{2} \qquad (4.3)
\end{aligned}$$

Multiply both sides of Equation 4.3 by $(D_\mu\mathbf{Q})^T$ and apply one of the constraints:

$$\begin{aligned}
(D_\mu\mathbf{Q})^T\check{\boldsymbol{\rho}} &= (D_\mu\mathbf{Q})^T\mathbf{1} + (D_\mu\mathbf{Q})^T\mathbf{Q}\frac{\lambda_0}{2} + (D_\mu\mathbf{Q})^T\mathbf{1}\frac{\lambda_1}{2} \\
(D_\mu\mathbf{Q})^T\check{\boldsymbol{\rho}} - (D_\mu\mathbf{Q})^T\mathbf{1} &= (D_\mu\mathbf{Q})^T\mathbf{Q}\frac{\lambda_0}{2} + (D_\mu\mathbf{Q})^T\mathbf{1}\frac{\lambda_1}{2} \\
\boldsymbol{\pi}^T\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q} &= \boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q})\frac{\lambda_0}{2} + \boldsymbol{\mu}^T\mathbf{Q}\frac{\lambda_1}{2} \qquad (4.4)
\end{aligned}$$

Multiply both sides of Equation 4.3 by $(D_\mu\mathbf{1})^T$ and apply the other constraint:

$$\begin{aligned}
(D_\mu\mathbf{1})^T\check{\boldsymbol{\rho}} &= (D_\mu\mathbf{1})^T\mathbf{1} + (D_\mu\mathbf{1})^T\mathbf{Q}\frac{\lambda_0}{2} + (D_\mu\mathbf{1})^T\mathbf{1}\frac{\lambda_1}{2} \\
(D_\mu\mathbf{1})^T\check{\boldsymbol{\rho}} - (D_\mu\mathbf{1})^T\mathbf{1} &= (D_\mu\mathbf{1})^T\mathbf{Q}\frac{\lambda_0}{2} + (D_\mu\mathbf{1})^T\mathbf{1}\frac{\lambda_1}{2} \\
0 &= \boldsymbol{\mu}^T\mathbf{Q}\frac{\lambda_0}{2} + \frac{\lambda_1}{2} \qquad (4.5)
\end{aligned}$$

We can now use Equations 4.4 and 4.5 to solve for the Lagrange multipliers:

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) & \boldsymbol{\mu}^T\mathbf{Q} \\ \boldsymbol{\mu}^T\mathbf{Q} & 1 \end{bmatrix} \begin{bmatrix} \frac{\lambda_0}{2} \\ \frac{\lambda_1}{2} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\pi}^T\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q} \\ 0 \end{bmatrix} \\
\begin{bmatrix} \frac{\lambda_0}{2} \\ \frac{\lambda_1}{2} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) & \boldsymbol{\mu}^T\mathbf{Q} \\ \boldsymbol{\mu}^T\mathbf{Q} & 1 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\pi}^T\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q} \\ 0 \end{bmatrix} \\
\frac{\lambda_0}{2} &= \frac{\boldsymbol{\pi}^T\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) - (\boldsymbol{\mu}^T\mathbf{Q})^2} && (4.6) \\
\frac{\lambda_1}{2} &= -\boldsymbol{\mu}^T\mathbf{Q}\frac{\boldsymbol{\pi}^T\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) - (\boldsymbol{\mu}^T\mathbf{Q})^2} && (4.7)
\end{aligned}$$

Substituting Equations 4.6 and 4.7 back into Equation 4.3, we get:

$$\begin{aligned}
\check{\boldsymbol{\rho}} &= \mathbf{1} + \mathbf{Q}\frac{\boldsymbol{\pi}^T\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) - (\boldsymbol{\mu}^T\mathbf{Q})^2} - \boldsymbol{\mu}^T\mathbf{Q}\frac{\boldsymbol{\pi}^T\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) - (\boldsymbol{\mu}^T\mathbf{Q})^2} \\
\check{\boldsymbol{\rho}} &= \mathbf{1} + \frac{\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) - (\boldsymbol{\mu}^T\mathbf{Q})^2}(\boldsymbol{\pi}^T\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q})
\end{aligned}$$

Replacing the inner products with corresponding expectations gives us:

$$\check{\rho}_a = 1 + \frac{Q_a - \mathbb{E}_\mu[Q]}{\mathbb{E}_\mu[Q^2] - \mathbb{E}_\mu[Q]^2}(\mathbb{E}_\pi[Q] - \mathbb{E}_\mu[Q]), \tag{4.8}$$

where again, expectations are implicitly conditioned on state. We refer to $\check{\rho}$ as the *Sparho* (Sparrow) weights. Interestingly, the above expression has a form where an action-value is mean-centered and normalized by its variance (under μ) and then scaled by a measure of off-policyness. We emphasize that these weights are not learned or approximated. They are computed in closed-form given a state's behavior and target policy probabilities and its immediate action-values. Interestingly, the estimator does not directly work with the policy probabilities, and only uses expectations computed under them. This may be related to regression importance sampling (Hanna et al., 2018) in that a *most likely* behavior policy might be implicitly used. It is unclear if a direct connection can be made, however, as the above weights typically cannot be expressed as a ratio of two policies.

By construction, we have:

1. $\check{\rho} = \rho$ is the unique solution to the constraints when there are 2 or fewer actions.
2. $\check{\rho}Q \sim \mu$ is an unbiased estimator of $\mathbb{E}_\pi[Q]$. Even with inaccurate value estimates, by satisfying the constraints, it provides an immediately unbiased estimate for the current values.
3. $\text{Var}_\mu(\check{\rho}) \leq \text{Var}_\mu(\rho)$. Ordinary importance sampling weights are in the space of solutions which satisfy the constraints, and variance is explicitly minimized.
4. $\check{\rho}_a = \rho_a = 1$ when $\mu = \pi$. This trivially minimizes the objective when on-policy, maintaining strict generalization of the on-policy setting akin to importance sampling.

Despite being unbiased in estimating expected action-values, inaccuracies in the value estimates will still result in a biased estimate of the true expected return. This is akin to the bias introduced by temporal difference methods (Sutton, 1988) and does not incur additional bias when correcting sampled action-values (e.g., off-policy Sarsa(0)). Additional bias can be introduced if $\check{\rho}$ is computed with inaccurate values and is then used to correct sampled rewards. Also of note, it may be possible for the importance weights to be negative. This may seem counter-intuitive in that it may lead to "undoing" updates, but should still give the correct expectation with lower variance. Nevertheless, we empirically found that the weights remained all positive the majority of the time, likely due to the second constraint tending solutions toward being positive. Lastly, computing $\check{\rho}$ requires first computing $\mathbb{E}_\pi[Q]$. It may seem redundant to compute this expectation only to form a sample-based estimator for the same quantity, but we argue that the benefit lies in the need to correct sequences, as opposed to only correcting immediate actions (Detailed in Section 4.3). We also emphasize that the importance weights presented are for one choice of objective. We explore alternative objectives in Appendix B, but focus on the above estimator due to its objective's relative simplicity.

4.2 The Sparho Estimator on Random Bandit Instances

In this section, we empirically evaluate the Sparho estimator in off-policy bandit prediction problems to gain intuition on the potential variance reduction and observe how relevant statistics scale with the action space. Bandit instances are randomly generated by sampling μ and π according to $\text{softmax}(\mathcal{N}_{|\mathcal{A}|\times 1}(0, \beta))$ and Q according to $\beta + \mathcal{N}_{|\mathcal{A}|\times 1}(0, \beta))$, with β tuning the determinism of the policies and the spread of the action-values. Using $\beta = 2$, we swept over action space sizes from 2 to 32,768. For each action space size, we generated 10,000 bandit instances and measured $\text{Var}(\rho Q)$ with ρ as the importance sampling and Sparho weights. We additionally compare clipped versions of each estimator akin to Retrace and Vtrace (Munos et al., 2016; Espeholt et al., 2018). For the clipped estimators, we additionally measure $\text{Bias}^2(\rho Q)$ and $\mathbb{E}_\mu[\rho]$. Results are shown in Figure 4.1.

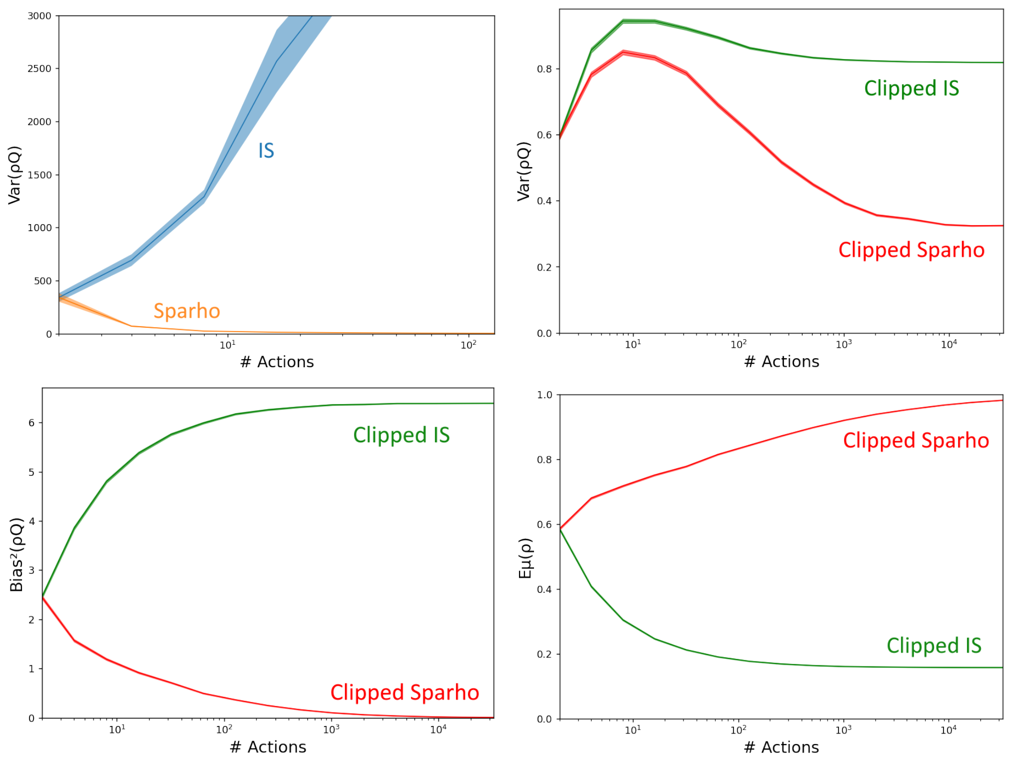


Figure 4.1: Various statistics against size of the action space, measured across randomly generated bandit instances. Results are averaged over 10,000 instances, and shaded regions represent one standard error.

It can be seen that the Sparho estimator has substantially lower variance than the importance sampling estimator. Perhaps surprisingly, we observe a *decrease* in variance with the size of the action space despite the exponential growth in that of the importance sampling estimator. With the clipped estimators, we see an overall substantial decrease in variance. However, the clipped Sparho estimator had lower variance, substantially lower bias, and maintained a larger expected importance weight than the clipped importance sampling estimator. With each metric, the clipped Sparho estimator similarly exhibited very favorable scaling with the size of the action space.

The above results assume access to the true action-values, computing relevant statistics in closed form for a random bandit instance. Below we consider the random bandit instances in the case where values are being learned. In addition to importance sampling, Sparho, and their clipped versions, we further evaluate regression importance sampling (Hanna et al., 2018) applied to both the importance sampling and Sparho estimators. We fix $|\mathcal{A}| = 8$ and $\alpha = 0.01$, and we note that the trends are reasonably consistent across the parameter space due to the simplicity of the problem. Averaged over 100 runs, we measure the absolute value error across 1,000 steps.

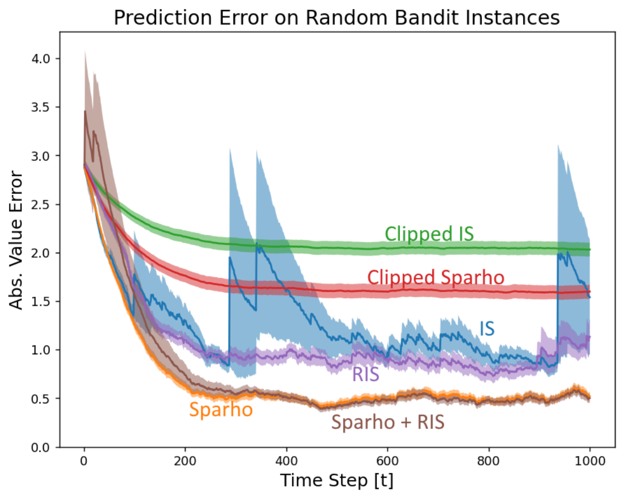


Figure 4.2: Absolute value error for various estimators across randomly generated bandit instances. Results are averaged over 100 instances, and shaded regions represent one standard error.

We see that even when action-values are being estimated, the Sparho estimator has substantially lower variance than the importance sampling counterpart for all variants tested (original, clipped, RIS). Given the use of a fixed step-size, this further leads to lower final value error as there is less variability to average out. We see general decrease in variance with the use of an estimated behavior policy for both RIS estimators. We further observe substantial bias in the solution of the clipped estimators, though the clipped Sparho estimator exhibited less of it.

4.3 Multi-step Corrections with Sparho Weights

In this section, we explore how Sparho weights might be used to correct a sequence. We start with the Bellman equation for q_π:

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\Big(r + \gamma \sum_{a'} \pi(a'|s')q_\pi(s',a')\Big). \tag{4.9}$$

We have that $\sum_a \pi(a|s)q_\pi(s,a) = \sum_a \mu(a|s)\check{\rho}(s,a)q_\pi(s,a)$, giving:

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\Big(r + \gamma \sum_{a'} \mu(a'|s')\check{\rho}(s',a')q_\pi(s',a')\Big). \tag{4.10}$$

Expanding the recursion multiple times and then drawing samples according to behavior policy μ, we get a corrected-return analogous to per-decision importance sampling (Precup et al., 2000):

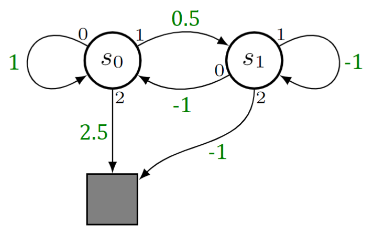
$$\begin{aligned}\hat{G}_t^{\check{\rho}} &= R_{t+1} + \gamma\check{\rho}_{t+1}\hat{G}_{t+1}^{\check{\rho}} \\ &= R_{t+1} + \gamma\check{\rho}_{t+1}R_{t+2} + \gamma^2\check{\rho}_{t+1}\check{\rho}_{t+2}R_{t+3} + \gamma^3\check{\rho}_{t+1}\check{\rho}_{t+2}\check{\rho}_{t+3}R_{t+4} + \dots \end{aligned} \tag{4.11}$$

Such a form, like importance sampling, conveniently lends itself to online, incremental multi-step algorithms like TD(λ). An important note, however, is that $\check{\rho}_k$ is computed based on the current value estimates for state S_k, and with inaccurate values, expanding the Bellman operator may give a quantity different from the values used to compute the importance weights. This suggests that additional bias may be introduced in this multi-step extension, but also that the true value function remains a fixed-point. It is unclear whether the resulting operator is still a contraction but we have not yet found an example with multiple fixed-points.

In the one-step case, a sample of Equation 4.10 gives Expected Sarsa, which is convergent (van

Seijen et al., 2009). In a loop-free episodic MDP, a state-action pair preceding a terminal state will be convergent as it only predicts the immediate reward. By induction, we get a cascade of one-step Expected Sarsa estimators with eventual stable bootstrap targets. It is uncertain whether we have such guarantees for general MDPs due to a non-linear dependence on Q, the existence of negative weights, and the non-stationarity of the weights as Q changes.

To try and understand the multi-step operator's behavior, we compute the *expected* every-visit Monte Carlo update for the per-decision Sparho estimator given by Equation 4.11 from various starting values in a two-state MDP. This lets us visualize update directions and gain intuition for the bias introduced in the Monte Carlo extreme of the multi-step case. We use a step size $\alpha = 0.1$, and show the importance sampling estimator for reference. Figure 4.3 precisely specifies the MDP, and Figure 4.4 visualizes the expected updates.



$$\gamma = 0.95$$

$$\pi = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{7}{10} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

$$\mu = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{10} & \frac{7}{3} & \frac{2}{3} \end{bmatrix}$$

Figure 4.3: 2-state MDP for visualizing expected multi-step Sparho updates.

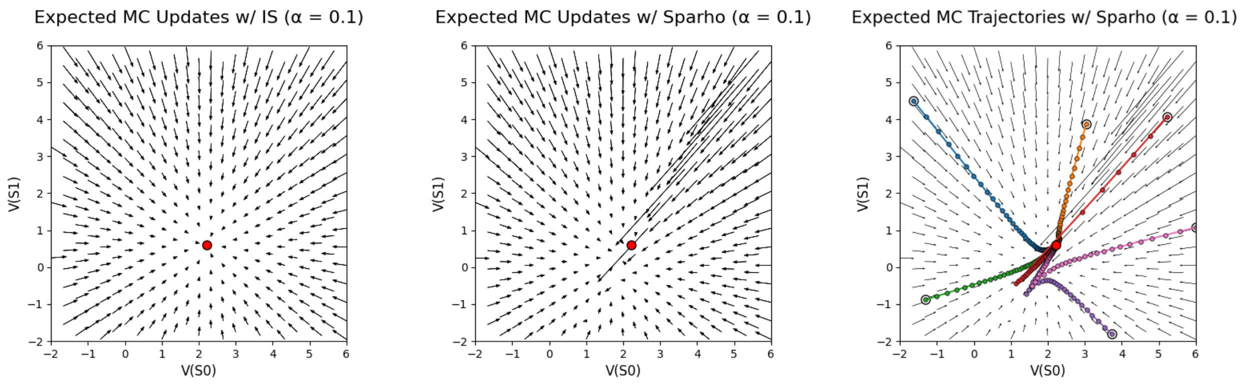


Figure 4.4: Expected every-visit Monte Carlo updates for the per-decision importance sampling and Sparho estimators in a 2-state MDP. The red circle denotes the fixed-point v_π, and sample trajectories of the Sparho estimator are visualized on the right.

As expected, the unbiased importance sampling estimator points directly at the fixed-point.The multi-step Sparho estimator generally has similar magnitude updates, but updates which point in slightly different directions relating to its bias. Visualizing trajectories, the values seem to rush to an area near the fixed-point before turning to approach it. Such paths may be an artifact of projecting higher-dimensional action-values into two state-values for visualization. Of possible concern are large jumps in the top right corner despite the small step size. We suspect that such large magnitude updates are due to interference between the transition matrix P_μ and the importance weights in the matrix $(I - \gamma P_\mu \text{diag}(\check{\rho}))^{-1}$. Under importance sampling, we have that $P_\pi = P_\mu \text{diag}(\rho)$, but this is not the case with $\check{\rho}$. It is difficult to comment on the generality and impact of this observation, but in stochastic empirical evaluation (See Section 4.4), such jumps appeared indistinguishable from typical return variance and did not appear to cause divergence.

With respect to the observed momentary large magnitude updates, we further note that multi-step TD estimators can attenuate them, e.g., n-step TD with per-decision Sparho weighting:

$$\begin{aligned}\hat{G}^{\check{\rho}}_{t:h} &= R_{t+1} + \gamma \check{\rho}_{t+1} \hat{G}^{\check{\rho}}_{t+1:h} \\ \hat{G}^{\check{\rho}}_{t:t} &= Q(S_t, A_t), \end{aligned} \tag{4.12}$$

where $h = t + n$. Such attenuation is expected as it interpolates Monte Carlo returns with the one-step extreme, which was previously acknowledged to be Expected Sarsa. Figure 4.5 explores this for $n \in \{4, 8, 16\}$ for the above MDP. Perhaps surprisingly, the 16-step TD update has no noticeable large jumps, and appears closer to the 4-step updates than Monte Carlo despite the MDP having an expected trajectory length of around 2 steps.

4.4 Extending and Evaluating Online Off-policy Algorithms

In this section, we empirically compare the multi-step Sparho estimator with importance sampling in a variety of environments. We specifically focus on the online, incremental setting with eligibility traces (Sutton, 1988; Precup et al., 2000).

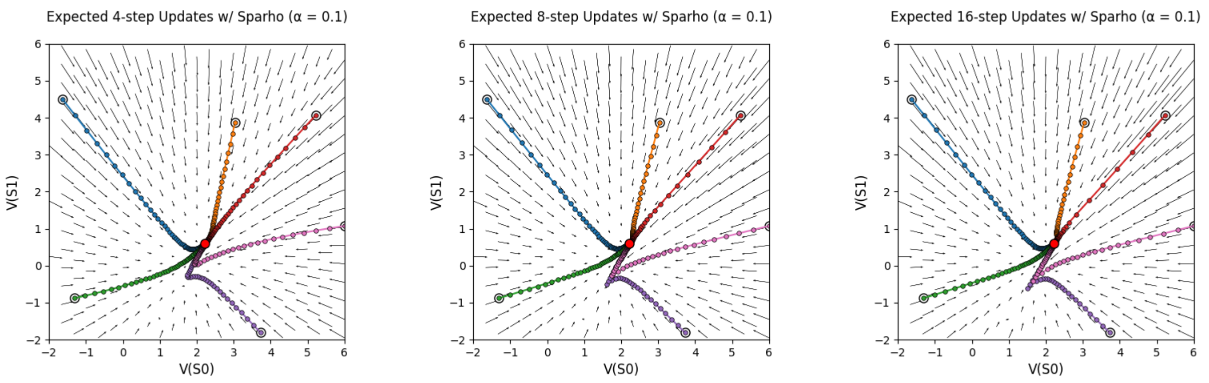


Figure 4.5: Expected n-step TD updates with per-decision Sparho weighting in a 2-state MDP. The red circle denotes the fixed-point v_π, with sample trajectories from a variety of starting points are visualized.

4.4.1 The Sparho(λ) Algorithm

We start with the following λ-return:

$$G_t^\lambda = R_{t+1} + \gamma\Big(\lambda\big(\rho_{t+1}G_{t+1}^\lambda + \mathbb{E}_\pi[Q(S_{t+1},\cdot)] - \rho_{t+1}Q(S_{t+1},A_{t+1})\big) + (1-\lambda)\mathbb{E}_\pi[Q(S_{t+1},\cdot)]\Big), \quad (4.13)$$

which can be re-written as a weighted sum of one-step Expected Sarsa TD errors:

$$G_t^\lambda = Q(S_t,A_t) + \sum_{k=t}^{\infty}\big(R_{k+1} + \gamma\mathbb{E}_\pi[Q(S_{k+1},\cdot)] - Q(S_k,A_k)\big)\prod_{i=t+1}^{k}\gamma\lambda\rho_i. \quad (4.14)$$

We denote the incremental algorithm using the above λ-return as Q(λ). It corresponds with the n-step per-decision estimator given by Equation 4.11, but with $\mathbb{E}_\pi[Q(S_{t+1},\cdot)] - \rho_{t+1}Q(S_{t+1},A_{t+1})$ appended to every reward beyond the immediate one. The appended terms have zero expected value and can be viewed as control variates, leading to a doubly-robust estimator (Jiang and Li, 2016; Thomas and Brunskill, 2016; De Asis and Sutton, 2018). They are central to the presence of one-step *Expected Sarsa* TD errors in Equation 4.14. Without them, we would have Sarsa(λ) with per-decision importance sampling (Precup et al., 2000) (i.e., $\mathbb{E}_\pi[Q(S_{k+1},\cdot)] \to \rho_{k+1}Q(S_{k+1},A_{k+1})$) leading to additional variance on the order of what is presented in Figure 4.1. This choice of λ-return is in anticipation of an analogous Sparho estimator, which must compute expected action-values anyways and leads to a fairer comparison. We denote the Sparho variant of Q(λ) as Sparho(λ):

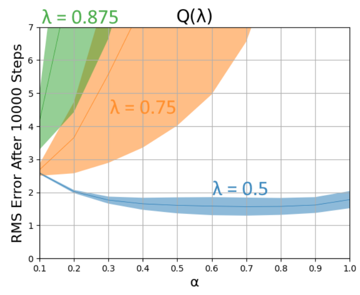
$$G_t^{\lambda,\check{\rho}} = Q(S_t, A_t) + \sum_{k=t}^{\infty} \big(R_{k+1} + \gamma \mathbb{E}_\pi[Q(S_{k+1}, \cdot)] - Q(S_k, A_k)\big) \prod_{i=t+1}^{k} \gamma\lambda\check{\rho}_i. \tag{4.15}$$

Additionally, we compare against clipped variants of each estimator, where the importance weights in the above λ-returns are clipped to the range $[0, 1]$. Clipping them in Equation 4.14 gives Retrace(λ) (Munos et al., 2016), and we denote the clipped variant of Equation 4.15 as ReSparho(λ).

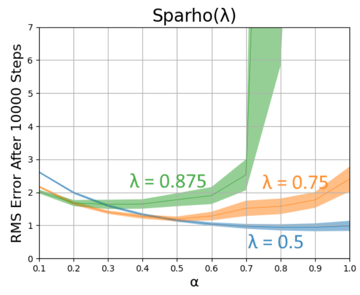
4.4.2 Tabular Multi-step Off-policy Prediction in a Loop-free MDP

We first evaluate the aforementioned algorithms in *Path World*, a loop-free episodic MDP. The environment is comparable to a fully-connected feed-forward neural network with one input and one output. Each neuron is a state and each connection is an action, with the input and output being the start state and terminal state, respectively. The network width represents the size of the action space, and the depth gives the number of decisions per episode. Each transition is assigned a fixed reward, and for each run, behavior and target policy probabilities are generated per-state with the procedure of Section 4.2's bandits with $\beta = 1$. We fix the depth to 5, consider $|\mathcal{A}| \in \{8, 32\}$, and performed a sweep over step size α and trace-decay rate λ. The root-mean-square (RMS) error between the learned and true values was measured after 10,000 steps for $|\mathcal{A}| = 8$ and after 500,000 steps for $|\mathcal{A}| = 32$. We performed 30 independent runs, and the results can be seen in Figure 4.6.

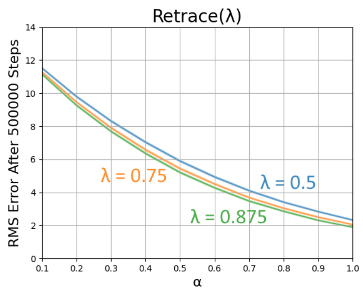
It can be seen that Q(λ) with 8 actions suffers from large variance with $\lambda > 0.5$ and struggles further with 32 actions. In contrast, Sparho(λ)'s reduced variance enables the use of larger λ with some instability for large (α, λ) pairs. For the clipped variants, we see substantially reduced variance, but slower learning due to the additional bias from an expected importance weight < 1. ReSparho(λ) outperforms Retrace(λ) across most parameter settings, consistent with the bandit results demonstrating a larger expected importance weight, but exhibits similar instability with larger (α, λ) pairs. Such instability may be due to the potential momentary large-magnitude updates observed in Figure 4.4, from the interference between P_μ and diag($\check{\rho}$).



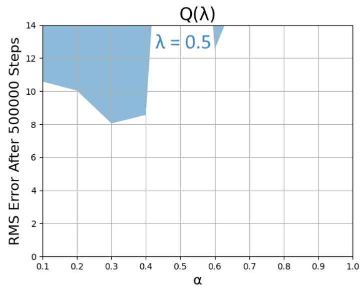
(a) Q(λ) with 8 actions



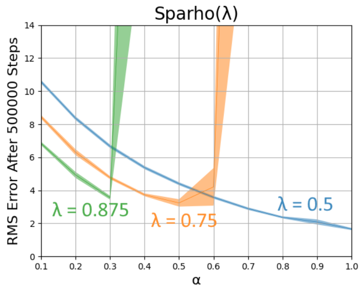
(b) Sparho(λ) with 8 actions



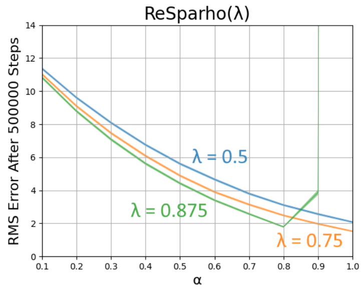
(c) Retrace(λ) with 32 actions



(d) Q(λ) with 32 actions



(e) Sparho(λ) with 32 actions



(f) ReSparho(λ) with 32 actions

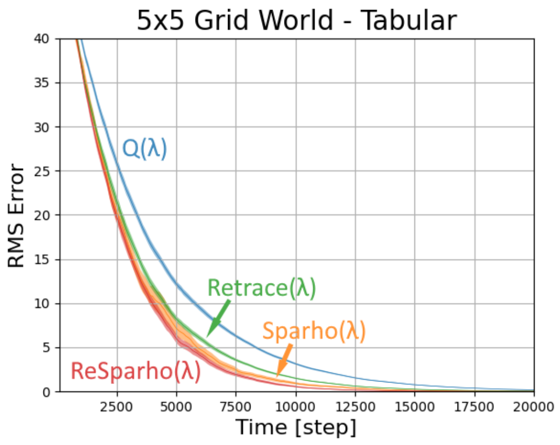
Figure 4.6: Results of an ablation study in the Path World environment for $|\mathcal{A}| \in \{8, 32\}$. Results are averaged over 30 runs, and shaded regions represent one standard error.

4.4.3 Tabular and Linear Multi-step Off-policy Prediction in a Grid World

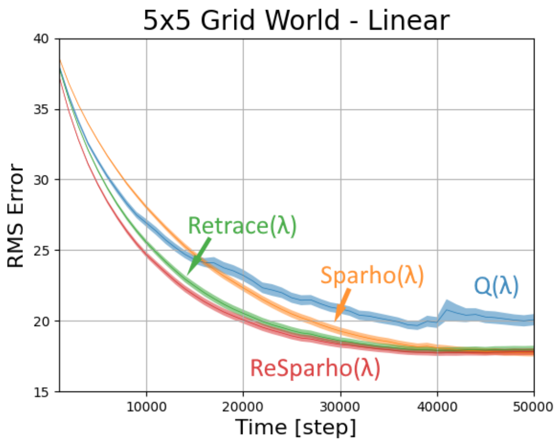
Next, we consider a 5×5 grid world with the center as the start state and two opposite corners as terminal states. Moving off of the grid keeps the agent in place, and a reward of -1 is received at each step. We first use a tabular representation with 4-directional movement, and we consider behavior and target policies which favor a specified action with probability $1-\epsilon$ and behave uniformly random otherwise. In contrast with the Path World environment, this MDP sees considerably many state revisitations within an episode. Performing a sweep over step size and trace-decay rate, we present learning curves for the best parameter setting in terms of RMS error after 20,000 steps. Results are averaged across 100 runs, and can be seen in Figure 4.7a.

To see how results scale to more difficult problems, we use an 8-directional-movement variant of the environment, and use feature vectors for use with linear function approximation. For each run, a random 16-bit binary feature vector is generated per state consisting of exactly 8 ones. This prevents perfectly representing the true values and allows us to see the potential bias the multi-step

Sparho operator may introduce in the linear fixed-point. A similar parameter sweep was performed, and learning curves are presented for the best parameter setting in terms of RMS error after 50,000 steps. Results are averaged across 100 runs and can be seen in Figure 4.7b.



(a) Tabular 5×5 Grid World results



(b) Linear 5×5 Grid World results

Figure 4.7: Results in the 5×5 Grid World environment with tabular and linear representations. Results are averaged over 100 runs, and shaded regions represent one standard error.

With the tabular representation, we see that Q(λ) was slowest to converge. Retrace(λ) substantially improves over it, with both Sparho(λ) and ReSparho(λ) performing better. Improved performance here largely comes from the ability to tolerate larger λ without becoming unstable and explains the apparent higher variance in the curves for the Sparho estimators. With the linear representation, we see a similar trend, but with Retrace(λ) learning quicker than Sparho(λ). Perhaps surprisingly, despite the aformentioned bias introduced by the multi-step Sparho operator, there is no noticeable impact on the final RMS error achieved— every algorithm but Q(λ) converges to a similar distance to the true value function.

4.4.4 Emphatic TD(λ) in the 5×5 Grid World

Here we evaluate the substitution of the Sparho estimator in Emphatic TD(λ). While we did not consider state-visitation distribution corrections in Sparho's derivation, such results can demonstrate whether Sparho weights are compatible with algorithms which re-weight updates on a state basis. We specifically evaluate EQ(λ) with importance sampling, Sparho, as well as their clipped versions, in the previous section's 5×5 Grid World environment with a tabular representation. We assume uniform interest $i(s) = 1, \forall s \in \mathcal{S}$, perform a sweep over step size α and trace-decay rate λ, and run each instance for 100,000 steps. Results are averaged over 100 runs, and the best parameter settings among the sweep in terms of final RMS error are shown in Figure 4.8.

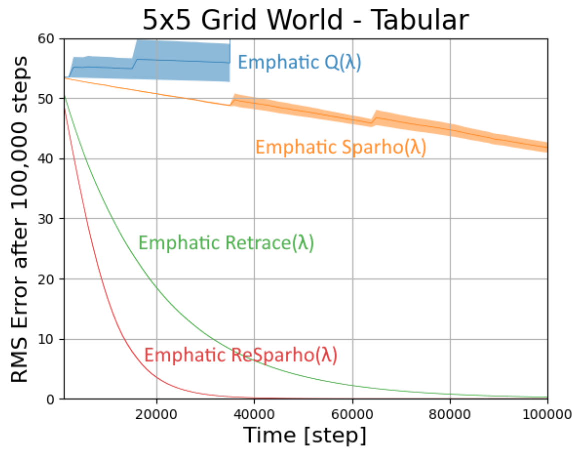


Figure 4.8: Results in the 5×5 Grid World environment with a tabular representation for Emphatic Q(λ) with various choices of importance weights. Results are averaged over 100 runs, and shaded regions represent one standard error.

We observe that despite the relative simplicity of the problem (a small, structured grid world), Emphatic Q(λ) with importance sampling does not reliably learn. Substituting $\rho_t \to \check{\rho}_t$ improves performance, but slowly progresses due to the necessity of a small step size to handle the base algorithm's substantial variance. Clipping both estimators enables substantially better learning progress, but notably much slower than their non-emphatic counterparts in Figure 4.7a.

4.4.5 Deep Multi-step Off-policy Control

Lastly, we evaluate the algorithms in the *Acrobot-v1* environment (Brockman et al., 2016) to demonstrate whether previous results extend toward control problems with non-linear function approximation. We used an ϵ-greedy behavior policy where ϵ was annealed from 1 to 0.1 over 25,000 steps and a target policy which was deterministically greedy with respect to the current estimates. All of the considered algorithms are equivalent in the on-policy case, and as a result, we evaluate each algorithm based on the the first 25,000 steps where there is a larger degree of off-policyness. For each step, we record the mean of the last 10 episodes and measure the area under the curve. A 2-hidden-layer fully-connected network with *tanh* activations was used to represent the action-values, and we performed a sweep over step size, trace-decay rate, and hidden layer width. Averaged across 30 runs, the results for the best hidden layer width (512) can be seen in Figure 4.9.

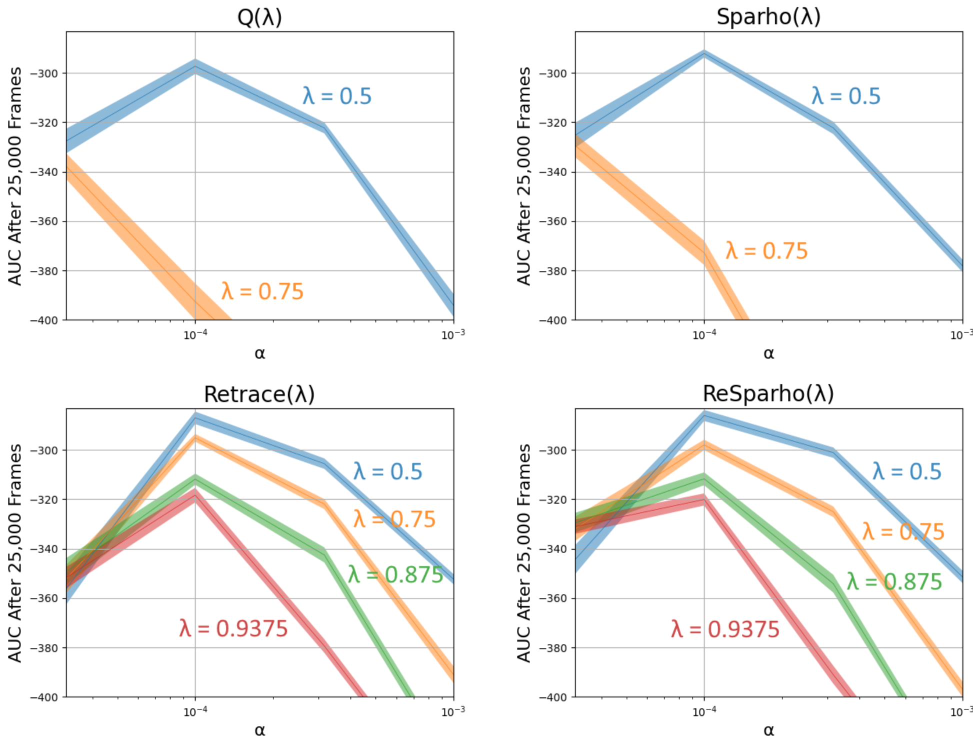


Figure 4.9: Results in the Acrobot-v1 environment with a hidden layer width of 512. Results are averaged over 30 runs, and shaded regions represent one standard error.

It can be seen that Sparho(λ) outperforms Q(λ), with lower variance in the episodic returns. ReSparho(λ) performs similar to Retrace(λ) *when properly tuned* but exhibits flatter sensitivity curves in the direction of smaller α. Similarities in performance are likely due to relatively small degrees of off-policyness (ϵ-greedy vs. greedy), a small action space, and how control performance might not require accurate value estimates. Nevertheless, all of the empirical evaluation taken together suggests that improved value estimation likely contributed to any improvements.

4.5 Summary

In this chapter, we investigated the use of value-aware importance weights as a lower variance alternative to importance sampling. We specifically derived a minimum-variance instance of such weights and empirically characterized the potential variance reduction. We then derived a per-decision multi-step estimator to allow for correcting sequences and analyzed the behavior of this operator in a small MDP. We then evaluated the online learning capabilities of our novel estimator, demonstrating improvements in off-policy prediction and control. Evaluation was done with tabular, linear, and non-linear function approximation, between pairs of algorithms where the sole difference is the substitution of value-aware importance weights.

Chapter 5

An Idiosyncrasy of Time-discretization in Reinforcement Learning

Many reinforcement learning algorithms assume that the agent-environment interaction occurs at synchronous, discrete time steps, where the environment waits for an action before advancing. In contrast, real-world physical systems are continuous in time and do not wait for an agent's input. As such, time-discretization becomes an important consideration (Mahmood et al., 2018a). Prior work suggests that current reinforcement learning algorithms are sensitive to the choice of discretization. Baird (1994) and Tallec et al. (2019) emphasize that action-values converge to state-values as the discretization interval approaches zero, creating degenerate cases for algorithms like Q-learning. Similarly, Munos (2006) show that the variance of policy gradients can explode under the same limit. Mahmood et al. (2018a) detail a trade-off between having fine-grained control and being able to discern the changes between subsequent states. Lastly, Farrahi and Mahmood (2023) provide practical guidelines for time-discretization-aware parameter selection by acknowledging how changes in discrete-time parameters influences an underlying continuous-time objective.

This chapter elaborates the dissertation's third area of contribution: identification and characterization of an idiosyncratic dependence on time discretization when naively applying a discrete-time algorithm to a discretized continuous-time environment. We view the discrete-time objective as a discrete approximation of the continuous-time objective. By considering *when* rewards occur, notably in typical setups for continuous-time control environments, the discrete-time return can be thought of as an atypical Riemann sum approximation to an underlying integral return.

5.1 Definitions of the Return

In discrete-time reinforcement learning (Sutton and Barto, 2018), the return from a time step t onward is defined to be:

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} R_{k+1}, \tag{5.1}$$

where again, T is the final time step in an episodic task, or ∞ in a continuing, infinite-horizon one. In continuous-time reinforcement learning (Doya, 2000; Mehta and Meyn, 2009; Frémaux et al., 2013; Lee and Sutton, 2021; Tallec et al., 2019), we instead define the *integral return*:

$$\tilde{G}_t = \int_t^T \gamma^{\tau-t} R_\tau d\tau. \tag{5.2}$$

This formulation is pertinent to applications with real-time interaction, e.g., robotics. Despite being continuous in time, robots are often digitally controlled, necessitating understanding the impact of the choice of time-discretization and how it relates these two objectives (Mahmood et al., 2018a).

5.2 When Rewards Occur

There are notation inconsistencies in the literature with respect to time indices in the discrete-time return (Equation 5.1). Some define it to start from R_{t+1} (Sutton, 1988; Precup et al., 2000; van Seijen et al., 2009; Barreto et al., 2017) as presented in this document, while some would start from R_t (Watkins, 1989; van Hasselt, 2010; Mnih et al., 2015; Wang et al., 2016). This inconsistency is inconsequential when solely considering the discrete-time setting as the rewards occur at the same locations in an agent's stream of experience. However, it has implications when viewed as a discrete approximation to an underlying integral return. Thus, it is worth considering *when* rewards occur.

We emphasize the focus on a setting where there is an underlying continuous-time objective, of which a digital learning agent samples at an arbitrary (and potentially variable) frequency. Despite the discrete-time notation inconsistencies, it is often agreed upon that from the agent's perspective, reward and next state are jointly observed. This is reflected in environment step calls in widely-used reinforcement learning APIs (Brockman et al., 2016), agent-environment interaction diagrams (Sutton and Barto, 2018), or explicit acknowledgement that reward can be a function of state, action, and *next state* (Puterman, 1994). In real-time settings which do not wait for an agent's input, meaningful evaluative feedback must come *after* time t as actions take time to execute and have a causal influence. Hardware limitations on sampling rates further impose an inherent delay in when a system can receive feedback for an action. In many existing robotics environments, where the considered setting is especially pertinent, rewards are often explicitly computed based on the next time step's state information (Todorov et al., 2012; Brockman et al., 2016; Mahmood et al., 2018b), e.g., rewards based on distance traveled in a direction between two time steps, or distance between an end-effector and a desired setpoint at the subsequent time step.

Of note, semi-MDPs and options (Sutton et al., 1999; Precup, 2000) address the problem of when rewards occur but under the assumption that one has access to higher-frequency interaction with the environment to integrate the discounted sum of rewards within a discretization interval. It is akin to the agent being aware of and able to time when each component of a temporally-extended reward occurs. Here, we consider when one *does not* have access to these higher-frequency samples, but is aware of how much time has elapsed between discrete decision points. Acquiring such information may not be possible due to hardware limitations, and highlights a nuance that arises when *naively* applying a discrete-time algorithm to a discretized continuous-time environment.

5.3 Implications for Time Discretization

If we consider rewards jointly arriving with the next state, at least from the agent's perspective, this results in an idiosyncrasy with respect to approximating an underlying integral return. While the discrete-time returns may use inconsistent reward time-indices, they are consistent on when discounting begins: the first reward is given weight $\gamma^0 = 1$, with subsequent rewards weighted by increasing powers of γ. We can view the integral return in Equation 5.2 to be of the form:

$$\int_t^T f(\tau)g(\tau)d\tau, \tag{5.3}$$

where $f(\tau)$ is the discounting term and $g(\tau)$ is the reward signal. A right-point Riemann sum approximation to this would yield:

$$\sum_{i=0}^{n-1} f(\tau_i)g(\tau_i)\Delta, \tag{5.4}$$

where $\Delta = \frac{T-t}{n}$ and $\tau = \{t+\Delta, t+2\Delta, ..., T\}$. The right-point Riemann sum beginning with $t+\Delta$ aligns with an agent jointly receiving a reward with the observation of the next state. However, this sum would weight the first reward by $\gamma^\Delta \neq \gamma^0$. This highlights that if one naively applies a discrete-time reinforcement learning algorithm to a discretized continuous-time environment, it is akin to evaluating the interval's left-point for discounting, and the right-point for rewards:

$$\sum_{i=0}^{n-1} f(\tau_i)g(\tau_{i+1})\Delta, \tag{5.5}$$

where $\tau \in \{t, t+\Delta, t+2\Delta, ..., T\}$. See Figure 5.1 for a visualization of this Riemann sum. This sum still converges to the correct integral when $n \to \infty$, as Bliss's Theorem (1914) allows each function to be evaluated at *any point* within the interval. However, for the case where a left-point Riemann sum is used for discounting, we expect this to perform *worse* than committing to a right-point Riemann sum. Due to the curvature of exponential decay, if one drew a rectangle with opposite corners at any two points, there will always be more area above the curve than below. That is, an underestimate has strictly lower error than an overestimate. This is visualized in Figure 5.2.

To rectify this discrepancy and commit to a right-point Riemann sum approximation, we simply multiply the discrete-time return by a factor of γ. For example, assuming $\Delta = 1$:

$$\gamma G_t = \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots$$

For a *fixed, pre-specified* action cycle-time Δ, there is no loss of generality as the discrete-time

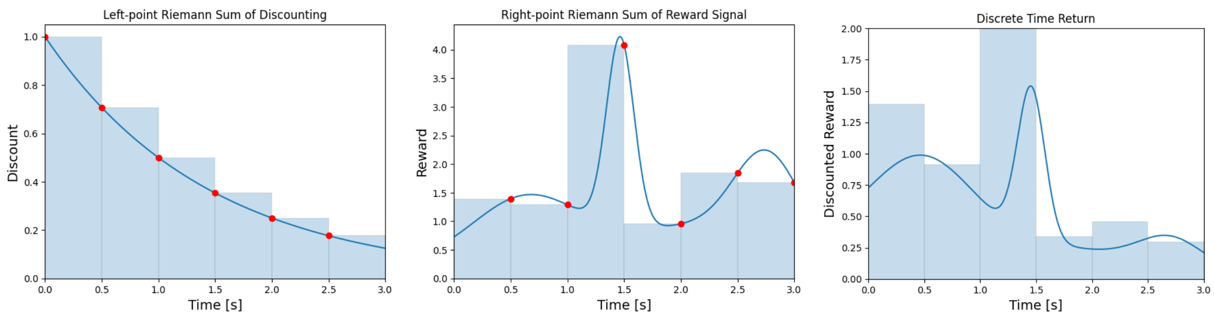


Figure 5.1: The resulting sum when applying a discrete-time algorithm to a discretized continuous-time domain. Note how rectangle heights may fall out of the function's range within an interval.

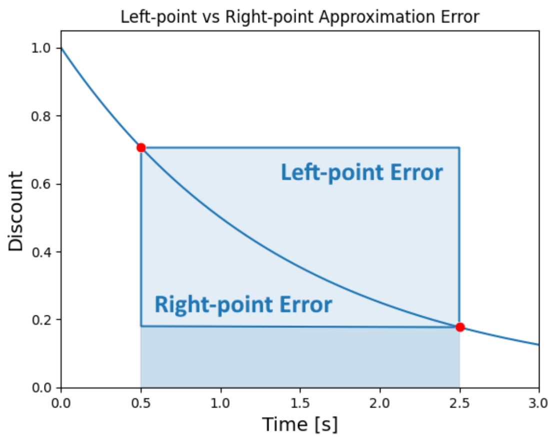


Figure 5.2: A visualization of the left-point and right-point Riemann sum approximation errors for an exponential decay. Due to curvature, a right-point Riemann sum will always have lower error.

return is proportional by a factor of $\gamma^{\Delta}\Delta$. However, it is unclear how this manifests when Δ is a tunable parameter, or may vary over time, e.g., due to an adaptive algorithm (Karimi et al., 2023) or inherent stochasticity. These concerns similarly apply to a tunable or variable γ in terms of a nuanced dependence on discretization. To emphasize the dependence on Δ, we note the explicit right-point Riemann sum return:

$$G_t^{RP} \overset{\text{def}}{=} \sum_{k=t}^{T-1} \gamma^{\sum_{i=t}^{k} \Delta_{i+1}} R_{k+1} \Delta_{k+1} \tag{5.6}$$

$$= \gamma^{\Delta_{t+1}} R_{t+1} \Delta_{t+1} + \gamma^{\Delta_{t+1}+\Delta_{t+2}} R_{t+2} \Delta_{t+2} + \cdots$$

Prior work has acknowledged the modifications of scaling rewards by Δ and exponentiating γ

by Δ in improving robustness to time-discretization (Tallec et al., 2019; Farrahi and Mahmood, 2023). The key difference and contribution in Equation 5.6 is the earlier discounting.

5.4 Comparison with Standard Riemann Sums

To see how the discrete-time return (DTR) in Equation 5.5 fares against a right-point Riemann sum, we evaluate them on randomly generated continuous-time signals. Inspired by robotics, we consider periodic signals and Gaussian mixtures. Periodic signals are comparable to signals pertaining to robot locomotion, while Gaussian mixtures instead resemble both sparse and distance-based rewards, depending on the spread of each Gaussian. We fix the signal length to 3 seconds with no loss of generality due to being continuous in time. Each signal generator is detailed below:

Random Periodic Signals - This signal sums 6 sinusoids $\sum_{i=0}^{5} A_i \sin(\omega_i t + \phi_i)$ with angular frequencies $\omega \in \{\frac{2\pi}{4}, \frac{2\pi}{2}, 2\pi, 4\pi, 8\pi, 16\pi\}$, amplitudes $A_i \sim \mathcal{N}(0,1)$, and phase shifts $\phi_i \sim \mathcal{U}(0, 2\pi)$.

Random Gaussian Mixtures - This signal sums 6 Gaussians $\sum_{i=0}^{5} \mathcal{N}(\mu_i, \sigma_i)$ with means $\mu_i \sim \mathcal{U}(0,3)$ and standard deviations $\sigma_i \sim \mathcal{U}(0, \frac{3}{2})$.

For each method, we varied the number of intervals $n \in \{5, 10, 25, 50, 100\}$, the discount factor $\gamma \in \{0.5, 0.75, 0.875\}$, and measured the absolute error of the integral approximation compared against a mid-point Riemann sum with 10^4 intervals. The values of γ used may appear small and unrepresentative of typical values. We however note that they discount *by seconds*, and that for a robot sampling every 30 ms, $\gamma = 0.5$ is effectively $\gamma^{\Delta} = 0.5^{0.03} \approx 0.98$ in discrete-time. Averaged across 10^6 randomly generated signals of each type, the results can be seen in Figure 5.3.

As expected, the errors generally increase as $\Delta \propto \frac{1}{n}$ increases. There is a consistent dip in error with the periodic signals, likely due to the intervals coincidentally aligning with the pre-specified frequencies. Across all settings, DTR had larger absolute error, consistent with our hypothesis that DTR would perform worse than right-point on discounted signals. The gap closes as $\gamma \to 1$, as the sums are equivalent at this extreme.

We then considered stochastic intervals to simulate variable time-discretization. This was im-

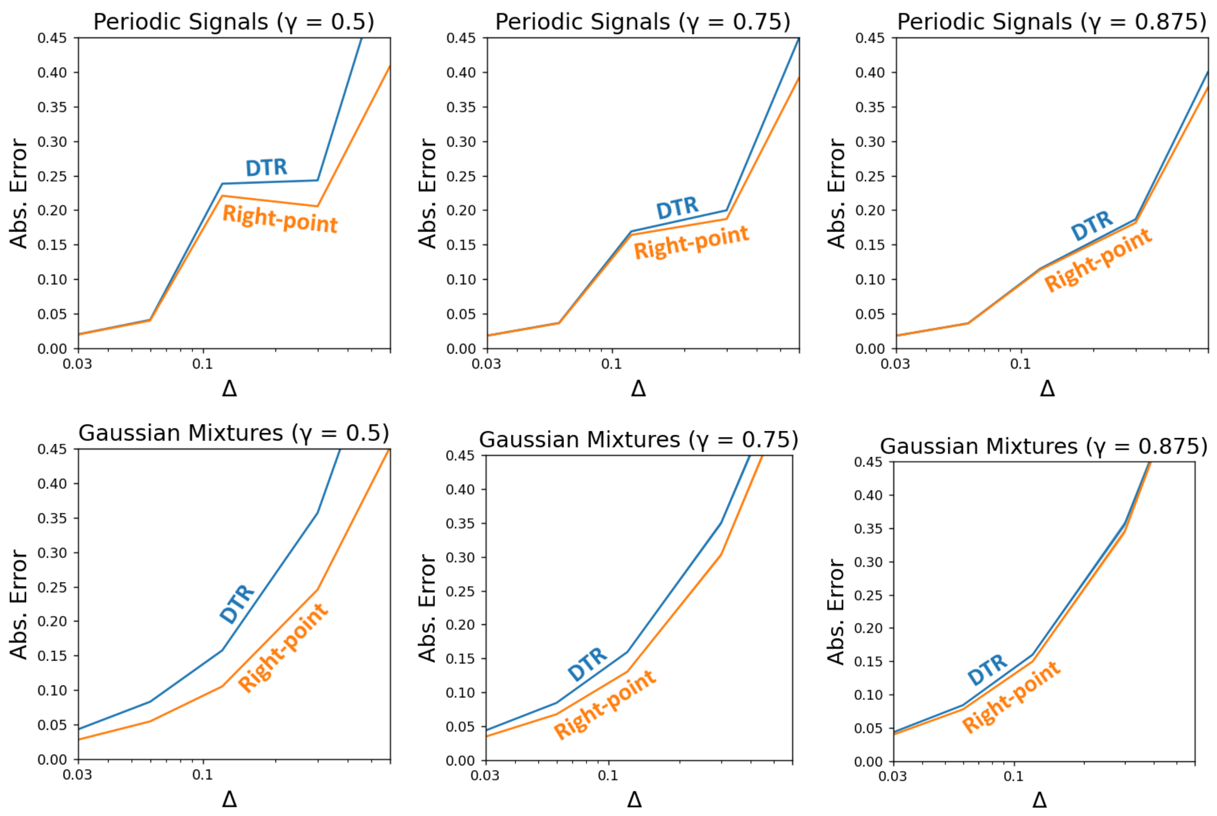


Figure 5.3: Numerical integration approximation error on *discounted* random signals. Results are averaged over 10^6 signals and shaded regions (too small for visibility) represent one standard error.

plemented by sampling, sorting, and re-scaling a set of $n + 1$ uniform random points representing interval endpoints. This is particularly pertinent as DTR is no longer proportional to right-point, and reflects the variability in applications on real-time systems. Fixing $\gamma = 0.75$, Figure 5.4 shows results averaged across 10^6 randomly generated signals of each type. Errors generally increased, with DTR maintaining larger approximation error across every setting.

Lastly, to see whether results hold beyond exponential discounting, we considered the product of each pair of the signal generators. This evaluates each sum in a more general numerical integration setting, while resembling variable, transition-dependent γ in reinforcement learning (White, 2016). Averaged across 10^6 randomly generated signal pairs, the results can be seen in Figure 5.5. Notably, the gap between DTR and the right-point Riemann sum widens dramatically. This suggests that beyond the structure of discounting, DTR is a generally worse integral approximation.

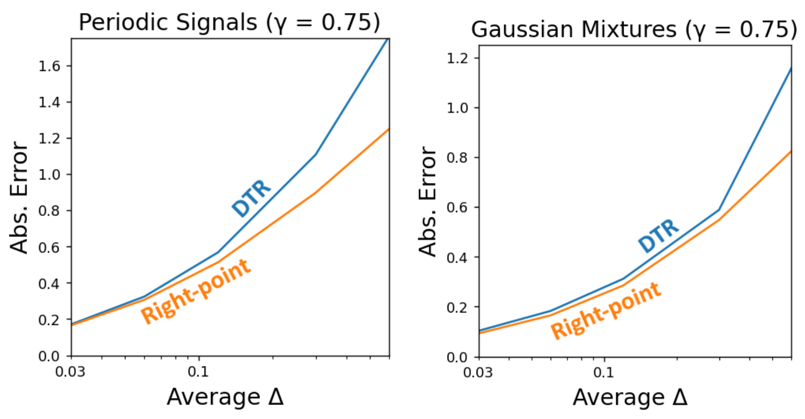


Figure 5.4: Numerical integration approximation error on *discounted* random signals, with stochastic discretization intervals. Results are averaged over 10^6 signals and shaded regions represent one standard error.

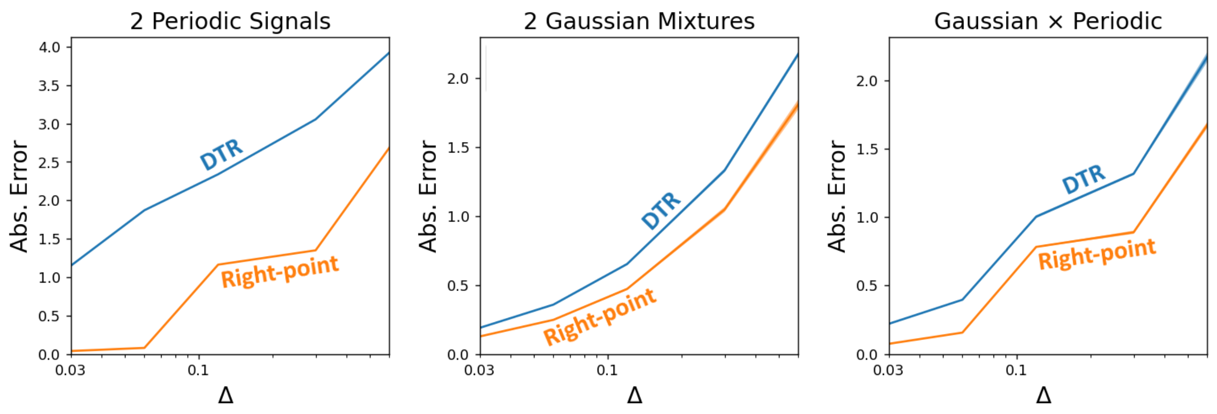


Figure 5.5: Numerical integration approximation error on *undiscounted* products of random signals. Results are averaged over 10^6 signals and shaded regions represent one standard error.

A key takeaway from these results is that shifting the discount factor in the discrete-time return yields a better prediction target (e.g., in value-based methods) in terms of error between the integral return. To reiterate, in the *fixed* Δ case, the sums are proportional despite the gaps in approximation error. This suggests that the improvement is inconsequential for control. However, such improved alignment is expected to improve control performance in the *variable* Δ setting, in terms of maximizing the underlying integral return. We explore this further in the next section.

5.5 Discretized Continuous-time Control

To evaluate the right-point Riemann sum in continuous-time control, we build off of the REINFORCE (Williams, 1992) algorithm. Despite this dissertation's focus on value-based methods, this choice is due to policy gradients being readily applicable to continuous actions and the algorithm's simplicity, allowing for better attribution of performance differences. We specifically use *online* REINFORCE with eligiblity traces (Kimura et al., 1995) and dropped γ^t term, summarized by:

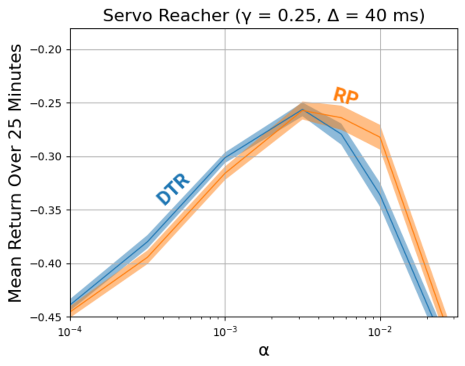
$$\begin{aligned}
&\mathbf{z} \leftarrow \mathbf{z} + \nabla_\theta \log \pi(A_t|S_t)\\
&\theta \leftarrow \theta + \alpha R_{\mathit{eff}}\mathbf{z}\\
&\mathbf{z} \leftarrow \gamma^{\Delta_{t+1}}\mathbf{z},
\end{aligned}$$

where Δ_{t+1} is the elapsed time between time steps t and $t+1$, $R_{\mathit{eff}} = R_{t+1}\Delta_{t+1}$ for the discrete-time return, and $R_{\mathit{eff}} = \gamma^{\Delta_{t+1}}R_{t+1}\Delta_{t+1}$ for the right-point Riemann sum. This algorithm employs Farrahi and Mahmood (2023)'s recommendations for making algorithms robust to time-discretization, emphasizing that the proposed right-point modification is complimentary. Each agent's policy was represented by a two-hidden-layer fully-connected network with *tanh* activations, and its output was treated as the mean of a Gaussian with an initial (bias unit) standard deviation of 1.

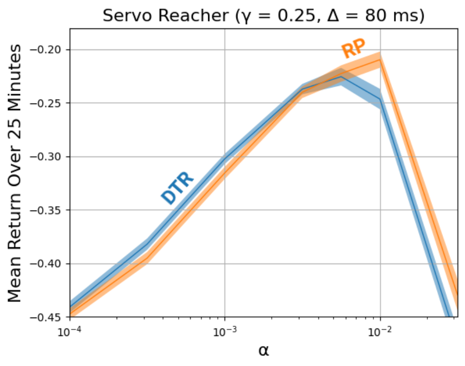
We designed a simulated *Servo Reacher* environment based on the setup in Mahmood et al. (2018b), with physical parameters sourced from a Dynamixel MX-28AT data sheet. This custom environment allows for fine-grained computation of the integral return, and flexibility in the discretization intervals an agent can sample at. Full environment specification can be found in Appendix C. To simulate the inherent stochasticity of a real robot, Gaussian noise was added to the target discretization interval, $\Delta_t \sim \mathcal{N}(\Delta_\mu, 10\text{ ms})$, with a hard minimum interval of 1 ms. We additionally included a 1% chance to sample the interval from $\mathcal{N}(1000\text{ ms}, 10\text{ ms})$ to simulate "catastrophic" events akin to communication errors. Of note, in less-exhaustive experiments not presented, such catastrophic events did not strongly impact or change the conclusions of the results.

The environment fixed $\gamma = 0.25$, where with an interval of 40 ms, corresponds with discrete-time

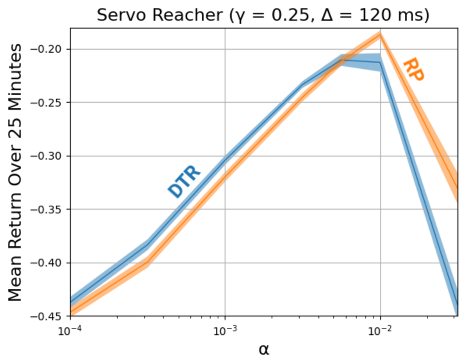
$\gamma^{0.04} \approx 0.95$. We considered target discretization intervals $\Delta_\mu \in \{40, 80, 120\}$ ms with a 4-second time limit, and measured the episodic integral return. Averaged over 100 runs of 25 (simulation) minutes, Figure 5.6 shows parameter sensitivity curves and the best parameters' learning curves.



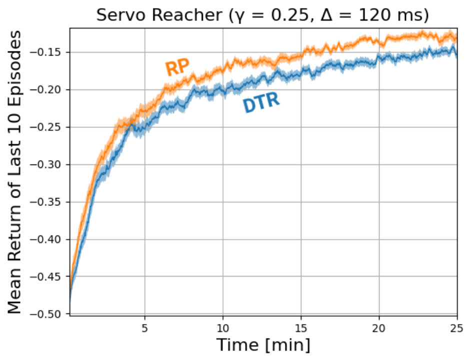
(a) Parameter Sensitivity ($\Delta_\mu = 40$ ms)



(b) Parameter Sensitivity ($\Delta_\mu = 80$ ms)



(c) Parameter Sensitivity ($\Delta_\mu = 120$ ms)



(d) Learning Curves ($\Delta_\mu = 120$ ms)

Figure 5.6: Servo Reacher results for REINFORCE using the discrete-time return (DTR) and right-point Riemann sum (RP), averaged over 100 runs. Shaded regions represent one standard error.

An initial observation is a systematic "lag" between the sensitivity curves of the two algorithms at low α. This is due to the return magnitudes being roughly proportional by a factor of $\mathbb{E}[\gamma^{\Delta_t}]$. If one absorbs this factor into the step-size, the right-point Riemann sum can be viewed as using a smaller *effective* α in the policy gradient update. Scaling the figure to use this effective α can be found to align the curves at low α. Nevertheless, we find that after accounting for this shift, REINFORCE with the right-point Riemann sum performs better for large α, and can significantly outperform the discrete-time return with both algorithms properly tuned. The right-point Riemann sum is seen to improve with *increasing* Δ_μ, in line with the approximation error results in Section

5.4. Acknowledging that the two returns are roughly proportional by $\mathbb{E}[\gamma^{\Delta_t}]$, improvements are expected as this term deviates from 1, i.e., decreasing γ or increasing Δ_μ.

5.6 Summary

In this chapter, we identified and characterized a nuance between the definitions of the discrete-time and continuous-time returns when viewing one as a discretization of the other. Our results suggest that when one is unable to evaluate the integral return via options, one can better align the objectives by shifting the discount factor to begin discounting sooner. This provides *unification* in that the discrete-time return becomes a relatively straight-forward discretization of the integral return. We strongly emphasize the modification's *simplicity*, and how apart from the $\gamma = 0$ extreme, the modification has no loss of generality when using fixed discretization intervals due to proportionality with the conventional discrete-time return. The returns are equivalent as $\gamma^{\Delta} \to 1$, but as they deviate, it results in better prediction targets in terms of integral approximation error and improves control performance when variable time-discretization is present. Beyond the integral approximation perspective, the modification has intuitive appeal in that results from catastrophically long delays are attenuated in the return, rather than fully crediting an action for that outcome.

Chapter 6

Concluding Remarks

Value-based methods form an important branch of reinforcement learning, where an agent constructs an understanding of the world through predicting long-term outcomes of its actions and then uses this information to inform decision-making. The GVF framework extends value function semantics to any signal of interest and highlights the promise of value-based methods for answering arbitrarily many predictive questions from a stream of experience. An objective of this dissertation was to better understand value-based methods and develop algorithmic ideas to improve them. As a result, I made a set of contributions toward value-based reinforcement learning:

1. I proposed the use of **fixed-horizon approximations** of the return to address instabilities when coupling off-policy bootstrapping with function approximation. The benefits stem from observing that non-recursive bootstrapping grounds predictions in a stable target— the zeroth horizon. I further provided better understanding of the predictive flexibility and benefits in using compositional GVFs, as this is a clear example where a collection of GVFs improves over a conventional value function, beyond the use of auxiliary tasks. I closed by providing an alternative way to derive behavior which capitalizes on the additional predictions available.

2. I introduced the idea of **value-aware importance weighting** as a framework for off-policy corrections, and I derived a minimum-variance instance of these weights to address variance concerns of importance sampling-based methods. I empirically characterized the potential

variance reduction and demonstrated its favorable scaling with the size of the action space. I later derived a per-decision multi-step estimator to extend these weights for multi-step corrections, and I explored the additional bias incurred by visualizing update trajectories in a small MDP. In such visualizations, I acknowledged potential momentary instability and how multi-step bootstrapping can address it. Through systematic empirical evaluation, I demonstrated improved value estimation over importance sampling-based counterparts.

3. I acknowledged an **idiosyncratic dependence on time-discretization** when naively applying a discrete-time algorithm to a discretized continuous-time environment. By considering *when* rewards occur, I argued that the discrete-time return poorly approximates the integral return compared with a conventional right-point Riemann sum. I showed how a simple modification to the discrete-time return implements a right-point Riemann sum, yielding improved prediction targets and improved control performance with variable time-discretization.

Taken together, this dissertation's contributions have produced improved value-based reinforcement learning algorithms and provided a deeper understanding of some key issues they face. Looking ahead, the foundational work laid here opens many avenues for future research:

1. I presented a form of weight sharing in extending FHTD methods to the deep reinforcement learning setting. Acknowledging that later horizons' value functions tend to be more similar than those ofldja earlier horizons, there may exist better architectures in support of the representational needs of this collection of GVFs. It is additionally worth considering an explicit decoupled representation learning procedure as this better aligns with the arguments around non-recursive bootstrapping.

2. Using an ϵ-democratic policy to derive behavior from an FHQ-learning agent demonstrated improved control performance. It would be worthwhile to formalize the intuitions around implicit iterative deepening, and better characterize what can be expected of such a policy. Further, one can explore other strategies for combining information from an ensemble of horizons and evaluate how this space of behavior policies scales to more complicated settings.

3. The multi-step per-decision Sparho estimator, while demonstrating good empirical performance, lacks theoretical understanding. I observed a potential for momentary instability, but the extent and implications of the issue are unclear. It would be fruitful to prove whether or not the estimator's fixed-point is unique and if the resulting algorithms are sound. Should issues be identified and characterized, it can inform how such weights can be derived to avoid them, e.g., by imposing additional constraints.

4. Sparho weights represent a specific instance of value-aware importance weights. However, the space of importance weights for off-policy corrections is vast that it warrants further exploration in terms of alternate objectives and constraints, e.g., using non-negative constraints or mixtures of Sparho and importance sampling weights to avoid momentary large updates.

5. The integral approximation perspective led to a right-point Riemann sum modification. However, if one were to additionally track a predecessor reward, it opens the possibility for other well-known integral approximations like the trapezoidal rule. Knowing the form of exponential discounting, we might also leverage a closed form integral of the discounting term, e.g., by weighting rewards by its mean-value over the interval.

A lot of recent work places emphasis on using deep neural networks to scale existing reinforcement learning algorithms for use in complex environments. Such work typically explores the extent of a classic algorithm's practical applicability or develops techniques specific to the use of neural networks. However, as evidenced by this dissertation's explorations, there remain plenty of questions at the fundamental level and I hope for this dissertation to inspire further work on the foundations of reinforcement learning. I personally subscribe to two trajectories for the field of reinforcement learning: (1) we discover a breakthrough in the fundamentals, after which things scale relatively smoothly with increasingly expressive function approximation, or (2) our current understanding and use of deep neural networks are insufficient for scaling reinforcement learning to complex problems, and we need an alternative framework for integrating expressive function approximation and deriving update rules. In both cases, we might benefit from taking a step back.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283.

Baird, L. (1994). Reinforcement learning in continuous time: advantage updating. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 2448–2453.

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the International Conference on Machine Learning.*

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., Silver, D., and van Hasselt, H. (2017). Successor Features for Transfer in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 4055–4065.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515.

Bertsekas, D. P. (2012). *Dynamic Programming & Optimal Control, Vol II: Approximate Dynamic Programming.* Athena Scientific, 4 edition.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming.* Athena Scientific, 1st edition.

Bliss, G. A. (1914). A Substitute for Duhamel's Theorem. *Annals of Mathematics*, 16(1/4):45–49.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. *CoRR*, abs/1606.01540.

De Asis, K., Chan, A., Pitis, S., Sutton, R. S., and Graves, D. (2020). Fixed-horizon temporal difference methods for stable reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, (04):3741–3748.

De Asis, K., Graves, E., and Sutton, R. S. (2023). Value-aware Importance Weighting for Off-policy Reinforcement Learning. In *Proceedings of the Conference on Lifelong Learning Agents*, pages 745–763.

De Asis, K. and Sutton, R. S. (2018). Per-decision multi-step temporal difference learning with control variates. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence.*

De Asis, K. and Sutton, R. S. (2024). An Idiosyncrasy of Time-discretization in Reinforcement Learning. In *Proceedings of the Reinforcement Learning Conference.*

Doya, K. (2000). Reinforcement Learning in Continuous Time and Space. *Neural Computation*, 12:219–245.

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

Farrahi, H. and Mahmood, A. R. (2023). Reducing the Cost of Cycle-Time Tuning for Real-World Policy Optimization. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8.

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

Frémaux, N., Sprekeler, H., and Gerstner, W. (2013). Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons. *PLOS Computational Biology*, 9:1–21.

Ghiassian, S., Patterson, A., White, M., Sutton, R. S., and White, A. (2018). Online off-policy prediction. *CoRR*, abs/1811.02597.

Graves, E. and Ghiassian, S. (2022). Importance Sampling Placement in Off-Policy Temporal-Difference Methods.

Hackman, L. (2012). *Faster Gradient-TD Algorithms.* Master's thesis, University of Alberta.

Hammersley, J. M. and Handscomb, D. C. (1964). *Monte Carlo methods.* Wiley, London; New York.

Hanna, J. P., Niekum, S., and Stone, P. (2018). Importance Sampling Policy Evaluation with an Estimated Behavior Policy. In *Proceedings of the International Conference on Machine Learning.*

Howard, R. A. (1960). *Dynamic Programming and Markov Processes.* MIT Press.

Jaakkola, T. S., Jordan, M. I., and Singh, S. P. (1994). On the Convergence of Stochastic Iterative Dynamic Programming Algorithms. *Neural Computation*, 6(6):1185–1201.

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

Jiang, N. and Li, L. (2016). Doubly Robust Off-policy Value Evaluation for Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning.*

Kahn, H. (1950a). Random sampling (Monte Carlo) techniques in neutron attenuation problems. I. *Nucleonics*, 6.

Kahn, H. (1950b). Random sampling (Monte Carlo) techniques in neutron attenuation problems. II. *Nucleonics*, 6.

Karimi, A., Jin, J., Luo, J., Mahmood, A. R., Jagersand, M., and Tosatto, S. (2023). Dynamic Decision Frequency with Continuous Options. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 7545–7552.

Kimura, H., Yamamura, M., and Kobayashi, S. (1995). Reinforcement Learning by Stochastic Hill Climbing on Discounted Reward. In *Proceedings of the International Conference on Machine Learning*.

Kloek, T. and Van Dijk, H. K. (1978). Bayesian estimates of equation system parameters: an application of integration by Monte Carlo. *Econometrica: Journal of the Econometric Society*, pages 1–19.

Lee, J. and Sutton, R. S. (2021). Policy iterations for reinforcement learning problems in continuous time and space — Fundamental theory and methods. *Automatica*, 126.

Li, L. and Littman, M. L. (2008). Efficient Value-Function Approximation via Online Linear Regression. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics*.

Littman, M., Sutton, R. S., and Singh, S. P. (2001). Predictive Representations of State. In *Advances in Neural Information Processing Systems*.

Liu, Y., Bacon, P.-L., and Brunskill, E. (2020). Understanding the curse of horizon in off-policy evaluation via conditional importance sampling. In *Proceedings of the International Conference on Machine Learning*.

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

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

Mahmood, A. R., Korenkevych, D., Komer, B. J., and Bergstra, J. (2018a). Setting up a Reinforcement Learning Task with a Real-World Robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4635–4640.

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. (2018b). Benchmarking Reinforcement Learning Algorithms on Real-World Robots. In *Proceedings of the Conference on Robot Learning.*

Mehta, P. G. and Meyn, S. P. (2009). Q-learning and Pontryagin's Minimum Principle. In *Proceedings of the IEEE Conference on Decision and Control*, pages 3598–3605.

Melo, F. S. and Ribeiro, M. I. (2007). Q-learning with linear function approximation. In *International Conference on Computational Learning Theory*, pages 308–322.

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., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Munos, R. (2006). Policy Gradient in Continuous Time. *Journal of Machine Learning Research*, 7:771–791.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. G. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems.*

Nachum, O., Chow, Y., Dai, B., and Li, L. (2019). DualDICE: Behavior-agnostic estimation of discounted stationary distribution corrections. *Advances in Neural Information Processing Systems.*

Ni, J. (2020). *Toward Emphatic Reinforcement Learning.* Master's thesis, University of Alberta.

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., and Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*.

Precup, D. (2000). *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts Amherst.

Precup, D., Sutton, R. S., and Singh, S. P. (2000). Eligibility traces for off-policy policy evaluation. In *Proceedings of the International Conference on Machine Learning*, pages 759–766.

Puterman, M. L. (1994). *Markov Decision Processes*. Wiley.

Riedmiller, M. (2005). Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. In *Proceedings of the European Conference on Machine Learning*, pages 317–328.

Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo Method*. Wiley, 1st edition.

Scherrer, B. (2010). Should one compute the Temporal Difference fix point or minimize the Bellman Residual? The unified oblique projection view. *CoRR*, abs/1011.4362.

Schlegel, M., White, A., Patterson, A., and White, M. (2021). General Value Function Networks. *Journal of Artificial Intelligence Research*, 70:497–543.

Sharifnassab, A. and Sutton, R. S. (2023). Toward Efficient Gradient-Based Value Estimation. In *Proceedings of the International Conference on Machine Learning*, pages 30827–30849.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Sozou, P. D. (1998). On Hyperbolic Discounting and Uncertain Hazard Rates. *Proceedings: Biological Sciences*, 265(1409):2015–2020.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* MIT Press, 2nd edition.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the International Conference on Machine Learning*, pages 993–1000.

Sutton, R. S., Mahmood, A. R., and White, M. (2016). An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning. *Journal of Machine Learning Research*, 17:73:1–73:29.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, pages 761–768.

Sutton, R. S., Precup, D., and 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. and Tanner, B. (2004). Temporal-Difference Networks. In *Advances in Neural Information Processing Systems.*

Tallec, C., Blier, L., and Ollivier, Y. (2019). Making Deep Q-learning methods robust to time discretization. In *Proceedings of the International Conference on Machine Learning*, pages 6096–6104.

Tanner, B. and Sutton, R. S. (2005). TD(λ) Networks: Temporal-Difference Networks with Eligibility Traces. In *Proceedings of the International Conference on Machine Learning*, pages 888–895.

Tesauro, G. (1995). TD-Gammon: A Self-Teaching Backgammon Program. *Applications of Neural Networks*, pages 267–285.

Thomas, P. and Brunskill, E. (2016). Data-Efficient Off-Policy Policy Evaluation for Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning.*

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. Technical report, IEEE Transactions on Automatic Control.

van Hasselt, H. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems.*

van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. (2018). Deep Reinforcement Learning and the Deadly Triad. *CoRR*, abs/1812.02648.

van Hasselt, H., Guez, A., Hessel, M., Mnih, V., and Silver, D. (2016). Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, pages 4287–4295.

van Hasselt, H. and Sutton, R. S. (2015). Learning to Predict Independent of Span. *CoRR*, abs/1508.04582.

van Seijen, H., Fatemi, M., and Tavakoli, A. (2019). Using a Logarithmic Mapping to Enable Lower Discount Factors in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 14111–14121.

van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., and Sutton, R. S. (2016). True Online Temporal-Difference Learning. *Journal of Machine Learning Research*, 17:145:1–145:40.

van Seijen, H., van Hasselt, H., Whiteson, S., and Wiering, M. A. (2009). A theoretical and empirical analysis of Expected Sarsa. In *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 177–184.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning*, pages 1995–2003.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College.

White, A. M. and White, M. (2016). Investigating practical, linear temporal difference learning. *CoRR*, abs/1602.08771.

White, M. (2016). Unifying task specification in reinforcement learning. *CoRR*, abs/1609.01995.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. In *IRE WESCON Convention Record, Part 4*, pages 96–104.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Wurman, P. R., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T. J., Capobianco, R., Devlic, A., Eckert, F., Fuchs, F., Gilpin, L., Khandelwal, P., Kompella, V., Lin, H., MacAlpine, P., Oller, D., Seno, T., Sherstan, C., Thomure, M. D., Aghabozorgi, H., Barrett, L., Douglas, R., Whitehead, D., Dürr, P., Stone, P., Spranger, M., and Kitano, H. (2022). Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228.

Zhang, R., Dai, B., Li, L., and Schuurmans, D. (2020). GenDICE: Generalized Offline Estimation of Stationary Values. In *Proceedings of the International Conference on Machine Learning*.

Appendix A Supplement for Chapter 3 (Fixed-horizon Temporal Difference Methods)

This appendix contains supplementary information for the contributions of Chapter 3 pertaining to fixed-horizon temporal difference methods.

A.1 FHTD(λ)

We can derive fixed-horizon TD(λ), denoted FHTD(λ), through this recursive form of its λ-return:

$$\hat{G}_t^{\lambda,h} = R_{t+1} + \gamma\Big((1-\lambda)V^{h-1}(S_{t+1}) + \lambda\hat{G}_{t+1}^{\lambda,h-1}\Big) \tag{A.1}$$

Assuming the values are not changing, we can get the following sum of one-step FHTD TD errors:

$$\delta_t^h = R_{t+1} + \gamma V^{h-1}(S_{t+1}) - V^h(S_t) \tag{A.2}$$

$$\hat{G}_t^{\lambda,h} = V^h(S_t) + \sum_{k=0}^{H-1} \delta_{t+k}^{h-k} \prod_{i=1}^{k} \gamma\lambda \tag{A.3}$$

Instead of storing rewards, it estimates the λ-return by backing up the current step's TD error, weighted by the product term in Equation A.3. Implementing this online and incrementally, this return can be computed given storage of the last H feature vectors, or via an eligibility trace (Sutton, 1988) which decays through horizon. An observation is that unlike n-step FHTD, FHTD(λ) requires learning all H value functions. Despite this, FHTD(λ) allows for *smooth* interpolation between one-step FHTD and fixed-horizon Monte Carlo while also trivially accommodating dynamically varying degrees of bootstrapping. We further note that alternative λ-return definitions (e.g., ones which mix carefully-selected n-step FHTD returns) may improve FHTD(λ)'s computational scaling.

A.2 Additional Experimental Details

Below are the hyperparameter settings considered in our DFHQ and DQN results. Bolded values represent DFHQ's best parameter combination in terms of average episodic return over 500,000 frames (area under the curve). DQN's best parameter combination was identical apart from a marginal improvement with target networks.

Parameter	**Value(s)**
Per Episode Frame Limit	5000
Replay Buffer Size	10^5
Mini-batch Size	32
RMSprop Learning Rate	$10^{-5}, \mathbf{10^{-4}}, 10^{-3}$
Hidden Layer Widths	$128, \mathbf{256}, 512, 4096$
DFHQ Final Horizon (H)	$32, \mathbf{64}$
Discount Rate (γ)	$\mathbf{0.99}, 1.0$
Target Net. Update Freq.	$\mathbf{1}, 100$

A.3 Algorithm Pseudocode

Algorithm 1 On-policy Linear One-step FHTD for estimating $V^H \approx v_\pi^H$

```
$\mathbf{w} \leftarrow$ Array of size $(H+1) \times m$
$\mathbf{w}_{[0]} \leftarrow$ [0 for $i$ in range($m$)]
$s \sim p(s_0)$
$a \sim \pi(\cdot|s)$
$t \leftarrow 0$
while $t \neq t_{max}$ do
    $s', r \sim p(s', r|s, a)$
    for $h = 1, 2, 3, ..., H$ do
        $\delta \leftarrow r + \gamma \mathbf{w}_{[h-1]} \cdot \phi(s') - \mathbf{w}_{[h]} \cdot \phi(s)$
        $\mathbf{w}_{[h]} \leftarrow \mathbf{w}_{[h]} + \alpha \delta \phi(s)$
    $s \leftarrow s'$
    $a \sim \pi(\cdot|s)$
    $t \leftarrow t + 1$
```

Algorithm 2 Off-policy Linear One-step FHQ-Learning for estimating $Q^H \approx q_*^H$

```
$\mathbf{w} \leftarrow$ Array of size $(H+1) \times m$
$\mathbf{w}_{[0]} \leftarrow [0$ for $i$ in range$(m)]$
$s \sim p(s_0)$
$a \sim \mu(\cdot|s)$ (e.g. $\epsilon$-greedy w.r.t. $Q^H(s,\cdot)$)
$t \leftarrow 0$
while $t \neq t_{max}$ do
    $s', r \sim p(s', r|s, a)$
    for $h = 1, 2, 3, ..., H$ do
        $\delta \leftarrow r + \gamma \max_{a'} \big(\mathbf{w}_{[h-1]} \cdot \phi(s', a')\big) - \mathbf{w}_{[h]} \cdot \phi(s, a)$
        $\mathbf{w}_{[h]} \leftarrow \mathbf{w}_{[h]} + \alpha\delta\phi(s, a)$
    $s \leftarrow s'$
    $a \sim \mu(\cdot|s)$
    $t \leftarrow t + 1$
```

Algorithm 3 On-policy Linear n-step FHTD for estimating $V^H \approx v_\pi^H$

```
$\mathbf{w} \leftarrow$ Array of size $(\frac{H}{n} + 1) \times m$
$\mathbf{w}_{[0]} \leftarrow [0$ for $i$ in range$(m)]$
$\mathbf{\Phi} \leftarrow$ Array of size $n \times m$
$\mathbf{R} \leftarrow$ Array of size $n \times 1$
$s \sim p(s_0)$
$a \sim \pi(\cdot|s)$
$t \leftarrow 0$
while $t \neq t_{max}$ do
    $\mathbf{\Phi}_{[t \pmod n]} = \phi(s)$
    $s', r \sim p(s', r|s, a)$
    $\mathbf{R}_{[t \pmod n]} = r$
    if $t + 1 \geq n$ then
        $r_{sum} \leftarrow$ discountedsum$(\mathbf{R})$
        $\phi_{old} \leftarrow \mathbf{\Phi}_{[(t+1-n) \pmod n]}$
        for $h_n = 1, 2, 3, ..., \frac{H}{n}$ do
            $\delta \leftarrow r_{sum} + \gamma^n \mathbf{w}_{[h_n - 1]} \cdot \phi(s') - \mathbf{w}_{[h_n]} \cdot \phi_{old}$
            $\mathbf{w}_{[h_n]} \leftarrow \mathbf{w}_{[h_n]} + \alpha\delta\phi_{old}$
    $s \leftarrow s'$
    $a \sim \pi(\cdot|s)$
    $t \leftarrow t + 1$
```

Algorithm 4 On-policy Linear FHTD(λ) for estimating $V^H \approx v_\pi^H$

```
$\mathbf{w} \leftarrow$ Array of size $(H+1) \times m$
$\mathbf{w}_{[0]} \leftarrow [0$ for $i$ in range$(m)]$
$\mathbf{\Phi} \leftarrow$ Array of size $H \times m$
$s \sim p(s_0)$
$a \sim \pi(\cdot|s)$
$t \leftarrow 0$
while $t \neq t_{max}$ do
    $\mathbf{\Phi}_{[t \pmod H]} \leftarrow \phi(s)$
    $s', r \sim p(s', r|s, a)$
    for $h = 1, 2, 3, ..., H$ do
        $\delta^h = r + \gamma \mathbf{w}_{[h-1]} \cdot \phi(s') - \mathbf{w}_{[h]} \cdot \phi(s)$
        for $i = 0, 1, 2, ..., H - h$ do
            $\mathbf{w}_{[h+i]} \leftarrow \mathbf{w}_{[h+i]} + \alpha(\gamma\lambda)^i \delta^h \mathbf{\Phi}_{[(t-i) \pmod H]}$
    $s \leftarrow s'$
    $a \sim \pi(\cdot|s)$
    $t \leftarrow t + 1$
```

Algorithm 5 Off-policy Linear FHQ(λ) for estimating $Q^H \approx q_*^H$

```
$\mathbf{w} \leftarrow$ Array of size $(H+1) \times m$
$\mathbf{w}_{[0]} \leftarrow [0$ for $i$ in range$(m)]$
$\mathbf{\Phi} \leftarrow$ Array of size $H \times m$
$\Pi \leftarrow$ Array of size $H \times H$
$s \sim p(s_0)$
$a \sim \mu(\cdot|s)$ (e.g. $\epsilon$-greedy w.r.t. $Q^H(s, \cdot)$)
$t \leftarrow 0$
while $t \neq t_{max}$ do
    $\mathbf{\Phi}_{[t \pmod H]} \leftarrow \phi(s, a)$
    $\Pi_{[t \pmod H]} \leftarrow [\mathbb{1}_{a = \operatorname{argmax} Q^h(s, \cdot)}$ for $h \in \{1, ..., H\}]$
    $s', r \sim p(s', r|s, a)$
    for $h = 1, 2, 3, ..., H$ do
        $\delta^h = r + \gamma \max_{a'} \left(\mathbf{w}_{[h-1]} \cdot \phi(s', a')\right) - \mathbf{w}_{[h]} \cdot \phi(s, a)$
        $e \leftarrow 1$
        for $i = 0, 1, 2, ..., H - h$ do
            $\mathbf{w}_{[h+i]} \leftarrow \mathbf{w}_{[h+i]} + \alpha e \delta^h \mathbf{\Phi}_{[(t-i) \pmod H]}$
            if $i \neq H - h$ then
                $e \leftarrow e\gamma\lambda\Pi_{[(t-i) \pmod H), h+i-1]}$
    $s \leftarrow s'$
    $a \sim \mu(\cdot|s)$
    $t \leftarrow t + 1$
```

Appendix B Supplement for Chapter 4 (Value-aware Importance Weighting)

This appendix contains supplementary information for the contributions of Chapter 4 pertaining to value-aware importance weighting.

B.1 Additional Experimental Details

B.1.1 Path World Details

Given actions $a \in \{0, 1, 2, \ldots, |\mathcal{A}|\}$, rewards were specified by $r(s, a) = \frac{1+a}{|\mathcal{A}|}$.

The scope of the parameter sweep is detailed below:

Parameter	**Value(s)**
Discount Factor γ	1.0
Action Space Size $\lvert\mathcal{A}\rvert$	$8, 32$
Step Size α	$0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$
Trace Decay λ	$0.5, 0.75, 0.875$

B.1.2 5 $\times$ 5 Grid World Details

The scope of the parameter sweep for the tabular results are detailed below:

Parameter	**Value(s)**
Discount Factor γ	1.0
Action Space Size $\lvert\mathcal{A}\rvert$	4
Action Commitment ϵ_π	0.5
Action Commitment ϵ_μ	1.0
Step Size α	$0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$
Trace Decay λ	$0.5, 0.75, 0.875, 0.9375, 0.96875, 0.984375, 0.9921875$

The scope of the parameter sweep for the linear function approximation results are detailed below:

Parameter	**Value(s)**
Discount Factor γ	1.0
Action Space Size $\lvert\mathcal{A}\rvert$	8
Action Commitment ϵ_π	0.5
Action Commitment ϵ_μ	0.5
Step Size α	$0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0$
Trace Decay λ	$0.5, 0.75, 0.875, 0.9375$

B.1.3 Emphatic TD(λ) Details

The scope of the parameter sweep are detailed below:

Parameter	**Value(s)**
Discount Factor γ	1.0
Action Space Size $\lvert\mathcal{A}\rvert$	4
Action Commitment ϵ_π	0.5
Action Commitment ϵ_μ	1.0
Step Size α	$0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1$
Trace Decay λ	$0.5, 0.75, 0.875, 0.9375, 0.96875$

B.1.4 Acrobot-v1 Details

The scope of the parameter sweep is detailed below:

Parameter	**Value(s)**
Discount Factor γ	0.99
Time Limit	500
Hidden Layers	2
Hidden Layer Width	$32, 64, 128, 256, 512$
Step Size α	$10^{-4.5}, 10^{-4}, 10^{-3.5}, 10^{-3}$
Trace Decay λ	$0.5, 0.75, 0.875, 0.9375$
Initial ϵ	1.0
Final ϵ	0.1
Linear ϵ Decay Steps	25000

B.2 Derivations for Alternate Objectives

Below we provide importance weight derivations for alternate optimization objectives.

B.2.1 Minimum $\ell 2$ distance to 1

The following is the Sparho objective but without the constraint on the expected importance weight being 1. This can have lower variance importance weights which are often close to 1, but may arbitrarily attenuate or amplify a sequence in the multi-step case.

$$\mathcal{L}(\check{\rho}, \lambda) = \sum_a \mu_a(\check{\rho}_a - 1)^2 + \lambda\Big(\sum_a \pi_a Q_a - \sum_a \mu_a \check{\rho}_a Q_a\Big)$$

$$\mathcal{L}(\check{\boldsymbol{\rho}}, \lambda) = (\check{\boldsymbol{\rho}} - 1)^T D_\mu (\check{\boldsymbol{\rho}} - 1) + \lambda_0\big(\boldsymbol{\pi}^T \mathbf{Q} - (D_\mu \mathbf{Q})^T \check{\boldsymbol{\rho}}\big)$$

$$
\begin{aligned}
\nabla\mathcal{L}(\check{\boldsymbol{\rho}},\lambda) = 0 &= 2D_\mu\check{\boldsymbol{\rho}} - 2\boldsymbol{\mu} - D_\mu\mathbf{Q}\lambda \\
0 &= D_\mu\check{\boldsymbol{\rho}} - \boldsymbol{\mu} - D_\mu\mathbf{Q}\frac{\lambda}{2} \\
0 &= D_\mu^{-1}D_\mu\check{\boldsymbol{\rho}} - D_\mu^{-1}\boldsymbol{\mu} - D_\mu^{-1}D_\mu\mathbf{Q}\frac{\lambda}{2} \\
0 &= \check{\boldsymbol{\rho}} - \mathbf{1} - \mathbf{Q}\frac{\lambda}{2} \\
\check{\boldsymbol{\rho}} &= \mathbf{1} + \mathbf{Q}\frac{\lambda}{2} \\
(D_\mu\mathbf{Q})^T\check{\boldsymbol{\rho}} &= (D_\mu\mathbf{Q})^T\mathbf{1} + (D_\mu\mathbf{Q})^T\mathbf{Q}\frac{\lambda}{2} \\
\boldsymbol{\pi}^T\mathbf{Q} &= \boldsymbol{\mu}^T\mathbf{Q} + \boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q})\frac{\lambda}{2} \\
\frac{\lambda}{2} &= \frac{\boldsymbol{\pi}^T\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q})} \\
\check{\boldsymbol{\rho}} &= \mathbf{1} + \frac{\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q})}(\boldsymbol{\pi}^T\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q})
\end{aligned}
$$

$$
\check{\boldsymbol{\rho}} = \mathbf{1} + \frac{\mathbf{Q}}{\mathbb{E}_\mu[Q^2]}(\mathbb{E}_\pi[Q] - \mathbb{E}_\mu[Q])
$$

B.2.2 Minimum $\ell 2$ distance to c

This is a generalization of C.1 intended to have the weights tend toward a specific value c, but again without a hard constraint on the expected importance weight.

$$
\mathcal{L}(\check{\rho},\lambda) = \sum_a \mu_a(\check{\rho}_a - c)^2 + \lambda\Big(\sum_a \pi_a Q_a - \sum_a \mu_a\check{\rho}_a Q_a\Big)
$$

$$
\mathcal{L}(\check{\boldsymbol{\rho}},\lambda) = (\check{\boldsymbol{\rho}} - c)^T D_\mu(\check{\boldsymbol{\rho}} - c) + \lambda_0\big(\boldsymbol{\pi}^T\mathbf{Q} - (D_\mu\mathbf{Q})^T\check{\boldsymbol{\rho}}\big)
$$

$$
\begin{aligned}
\nabla\mathcal{L}(\check{\boldsymbol{\rho}},\lambda) = 0 &= 2D_\mu\check{\boldsymbol{\rho}} - 2c\boldsymbol{\mu} - D_\mu\mathbf{Q}\lambda \\
0 &= D_\mu\check{\boldsymbol{\rho}} - c\boldsymbol{\mu} - D_\mu\mathbf{Q}\frac{\lambda}{2} \\
0 &= D_\mu^{-1}D_\mu\check{\boldsymbol{\rho}} - D_\mu^{-1}c\boldsymbol{\mu} - D_\mu^{-1}D_\mu\mathbf{Q}\frac{\lambda}{2} \\
0 &= \check{\boldsymbol{\rho}} - \mathbf{c} - \mathbf{Q}\frac{\lambda}{2} \\
\check{\boldsymbol{\rho}} &= \mathbf{c} + \mathbf{Q}\frac{\lambda}{2} \\
(D_\mu\mathbf{Q})^T\check{\boldsymbol{\rho}} &= (D_\mu\mathbf{Q})^T\mathbf{c} + (D_\mu\mathbf{Q})^T\mathbf{Q}\frac{\lambda}{2} \\
\boldsymbol{\pi}^T\mathbf{Q} &= c\boldsymbol{\mu}^T\mathbf{Q} + \boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q})\frac{\lambda}{2} \\
\frac{\lambda}{2} &= \frac{\boldsymbol{\pi}^T\mathbf{Q} - c\boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q})} \\
\check{\boldsymbol{\rho}} &= \mathbf{c} + \frac{\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q})}(\boldsymbol{\pi}^T\mathbf{Q} - c\boldsymbol{\mu}^T\mathbf{Q})
\end{aligned}
$$

$$
\check{\boldsymbol{\rho}} = \mathbf{c} + \frac{\mathbf{Q}}{\mathbb{E}_\mu[Q^2]}(\mathbb{E}_\pi[Q] - c\mathbb{E}_\mu[Q])
$$

B.2.3 Minimum variance around constrained length c

This is a generalization of the minimum-variance Sparho objective, but with a variable expected importance weight magnitude. One possible application of this is to absorb the discount factor into the importance weights to simultaneously discount and correct the distribution of a sample.

$$
\begin{aligned}
\mathcal{L}(\check{\rho},\lambda_0,\lambda_1) &= \sum_a \mu_a(\check{\rho}_a - c)^2 + \lambda_0\Big(\sum_a \pi_a Q_a - \sum_a \mu_a\check{\rho}_a Q_a\Big) + \lambda_1\Big(c - \sum_a \mu_a\check{\rho}_a\Big) \\
\mathcal{L}(\check{\boldsymbol{\rho}},\lambda_0,\lambda_1) &= (\check{\boldsymbol{\rho}} - c)^T D_\mu(\check{\boldsymbol{\rho}} - c) + \lambda_0\big(\boldsymbol{\pi}^T\mathbf{Q} - (D_\mu\mathbf{Q})^T\check{\boldsymbol{\rho}}\big) + \lambda_1(c - (D_\mu\mathbf{1})^T\check{\boldsymbol{\rho}})
\end{aligned}
$$

$$\begin{aligned}
\nabla\mathcal{L}(\check{\boldsymbol{\rho}},\lambda_0,\lambda_1) = 0 &= 2D_\mu\check{\boldsymbol{\rho}} - 2c\boldsymbol{\mu} - D_\mu\mathbf{Q}\lambda_0 - D_\mu\mathbf{1}\lambda_1 \\
0 &= D_\mu\check{\boldsymbol{\rho}} - c\boldsymbol{\mu} - D_\mu\mathbf{Q}\frac{\lambda_0}{2} - D_\mu\mathbf{1}\frac{\lambda_1}{2} \\
0 &= D_\mu^{-1}D_\mu\check{\boldsymbol{\rho}} - D_\mu^{-1}c\boldsymbol{\mu} - D_\mu^{-1}D_\mu\mathbf{Q}\frac{\lambda_0}{2} - D_\mu^{-1}D_\mu\mathbf{1}\frac{\lambda_1}{2} \\
0 &= \check{\boldsymbol{\rho}} - \mathbf{c} - \mathbf{Q}\frac{\lambda_0}{2} - \mathbf{1}\frac{\lambda_1}{2} \\
\check{\boldsymbol{\rho}} &= \mathbf{c} + \mathbf{Q}\frac{\lambda_0}{2} + \mathbf{1}\frac{\lambda_1}{2}
\end{aligned}$$

$$\begin{aligned}
(D_\mu\mathbf{Q})^T\check{\boldsymbol{\rho}} &= (D_\mu\mathbf{Q})^T\mathbf{c} + (D_\mu\mathbf{Q})^T\mathbf{Q}\frac{\lambda_0}{2} + (D_\mu\mathbf{Q})^T\mathbf{1}\frac{\lambda_1}{2} \\
(D_\mu\mathbf{Q})^T\check{\boldsymbol{\rho}} - (D_\mu\mathbf{Q})^T\mathbf{c} &= (D_\mu\mathbf{Q})^T\mathbf{Q}\frac{\lambda_0}{2} + (D_\mu\mathbf{Q})^T\mathbf{1}\frac{\lambda_1}{2} \\
\boldsymbol{\pi}^T\mathbf{Q} - c\boldsymbol{\mu}^T\mathbf{Q} &= \boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q})\frac{\lambda_0}{2} + \boldsymbol{\mu}^T\mathbf{Q}\frac{\lambda_1}{2}
\end{aligned}$$

$$\begin{aligned}
(D_\mu\mathbf{1})^T\check{\boldsymbol{\rho}} &= (D_\mu\mathbf{1})^T\mathbf{c} + (D_\mu\mathbf{1})^T\mathbf{Q}\frac{\lambda_0}{2} + (D_\mu\mathbf{1})^T\mathbf{1}\frac{\lambda_1}{2} \\
(D_\mu\mathbf{1})^T\check{\boldsymbol{\rho}} - (D_\mu\mathbf{1})^T\mathbf{c} &= (D_\mu\mathbf{1})^T\mathbf{Q}\frac{\lambda_0}{2} + (D_\mu\mathbf{1})^T\mathbf{1}\frac{\lambda_1}{2} \\
0 &= \boldsymbol{\mu}^T\mathbf{Q}\frac{\lambda_0}{2} + \frac{\lambda_1}{2}
\end{aligned}$$

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) & \boldsymbol{\mu}^T\mathbf{Q} \\ \boldsymbol{\mu}^T\mathbf{Q} & 1 \end{bmatrix} \begin{bmatrix} \frac{\lambda_0}{2} \\ \frac{\lambda_1}{2} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\pi}^T\mathbf{Q} - c\boldsymbol{\mu}^T\mathbf{Q} \\ 0 \end{bmatrix} \\
\begin{bmatrix} \frac{\lambda_0}{2} \\ \frac{\lambda_1}{2} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) & \boldsymbol{\mu}^T\mathbf{Q} \\ \boldsymbol{\mu}^T\mathbf{Q} & 1 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\pi}^T\mathbf{Q} - c\boldsymbol{\mu}^T\mathbf{Q} \\ 0 \end{bmatrix} \\
\frac{\lambda_0}{2} &= \frac{\boldsymbol{\pi}^T\mathbf{Q} - c\boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) - (\boldsymbol{\mu}^T\mathbf{Q})^2} \\
\frac{\lambda_1}{2} &= -\boldsymbol{\mu}^T\mathbf{Q}\frac{\boldsymbol{\pi}^T\mathbf{Q} - c\boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) - (\boldsymbol{\mu}^T\mathbf{Q})^2}
\end{aligned}$$

$$\check{\boldsymbol{\rho}} = \mathbf{c} + \mathbf{Q}\frac{\boldsymbol{\pi}^T\mathbf{Q} - c\boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) - (\boldsymbol{\mu}^T\mathbf{Q})^2} - \boldsymbol{\mu}^T\mathbf{Q}\frac{\boldsymbol{\pi}^T\mathbf{Q} - c\boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) - (\boldsymbol{\mu}^T\mathbf{Q})^2}$$
$$\check{\boldsymbol{\rho}} = \mathbf{c} + \frac{\mathbf{Q} - \boldsymbol{\mu}^T\mathbf{Q}}{\boldsymbol{\mu}^T(\mathbf{Q}\circ\mathbf{Q}) - (\boldsymbol{\mu}^T\mathbf{Q})^2}(\boldsymbol{\pi}^T\mathbf{Q} - c\boldsymbol{\mu}^T\mathbf{Q})$$

$$\check{\boldsymbol{\rho}} = \mathbf{c} + \frac{\mathbf{Q} - \mathbb{E}_\mu[Q]}{\mathbb{E}_\mu[Q^2] - \mathbb{E}_\mu[Q]^2}(\mathbb{E}_\pi[Q] - c\mathbb{E}_\mu[Q])$$

B.2.4 Minimum variance of $\check{\rho}Q$

This objective aims to minimize the variance of $\check{\rho}Q$. Such an objective has intuitive appeal in minimizing the variance of the full update target, as opposed to just the importance weight. However, despite the focus on value-awareness, such an objective may be too reliant on the specific value estimates, and very sensitive to deviations in them. This may lead to substantially more bias, and even an increase in variance should the value estimates not accurately reflect the return.

Let $V = \mathbb{E}_\pi[Q] = \sum_a \pi_a Q_a$:

$$\mathcal{L}(\check{\rho}, \lambda_0, \lambda_1) = \sum_a \mu_a(\check{\rho}_a Q_a - V)^2 + \lambda_0\Big(V - \sum_a \mu_a\check{\rho}_a Q_a\Big) + \lambda_1\Big(1 - \sum_a \mu_a\check{\rho}_a\Big)$$
$$\mathcal{L}(\check{\boldsymbol{\rho}}, \lambda_0, \lambda_1) = (D_Q\check{\boldsymbol{\rho}} - V)^T D_\mu(D_Q\check{\boldsymbol{\rho}} - V) + \lambda_0\big(V - (D_\mu\mathbf{Q})^T\check{\boldsymbol{\rho}}\big) + \lambda_1(1 - (D_\mu\mathbf{1})^T\check{\boldsymbol{\rho}})$$

$$\begin{aligned}
\nabla\mathcal{L}(\check{\boldsymbol{\rho}}, \lambda_0, \lambda_1) = 0 &= 2D_\mu D_Q D_Q\check{\boldsymbol{\rho}} - 2D_\mu\mathbf{Q}V - D_\mu\mathbf{Q}\lambda_0 - D_\mu\mathbf{1}\lambda_1 \\
0 &= D_\mu D_Q D_Q\check{\boldsymbol{\rho}} - D_\mu\mathbf{Q}V - D_\mu\mathbf{Q}\frac{\lambda_0}{2} - D_\mu\mathbf{1}\frac{\lambda_1}{2} \\
0 &= D_Q^{-1}D_Q^{-1}D_\mu^{-1}D_\mu D_Q D_Q\check{\boldsymbol{\rho}} - D_Q^{-1}D_Q^{-1}D_\mu^{-1}D_\mu\mathbf{Q}V - D_Q^{-1}D_Q^{-1}D_\mu^{-1}D_\mu\mathbf{Q}\frac{\lambda_0}{2} \\
&\quad - D_Q^{-1}D_Q^{-1}D_\mu^{-1}D_\mu\mathbf{1}\frac{\lambda_1}{2} \\
0 &= \check{\boldsymbol{\rho}} - D_Q^{-1}\mathbf{1}V - D_Q^{-1}\mathbf{1}\frac{\lambda_0}{2} - D_Q^{-1}D_Q^{-1}\mathbf{1}\frac{\lambda_1}{2} \\
\check{\boldsymbol{\rho}} &= D_Q^{-1}\mathbf{1}V + D_Q^{-1}\mathbf{1}\frac{\lambda_0}{2} + D_Q^{-1}D_Q^{-1}\mathbf{1}\frac{\lambda_1}{2}
\end{aligned}$$

$$
\begin{aligned}
(D_\mu \mathbf{Q})^T \check{\boldsymbol{\rho}} &= (D_\mu \mathbf{Q})^T D_Q^{-1} \mathbf{1} V + (D_\mu \mathbf{Q})^T D_Q^{-1} \mathbf{1} \frac{\lambda_0}{2} + (D_\mu \mathbf{Q})^T D_Q^{-1} D_Q^{-1} \mathbf{1} \frac{\lambda_1}{2} \\
(D_\mu \mathbf{Q})^T \check{\boldsymbol{\rho}} - (D_\mu \mathbf{Q})^T D_Q^{-1} \mathbf{1} V &= (D_\mu \mathbf{Q})^T D_Q^{-1} \mathbf{1} \frac{\lambda_0}{2} + (D_\mu \mathbf{Q})^T D_Q^{-1} D_Q^{-1} \mathbf{1} \frac{\lambda_1}{2} \\
0 &= \frac{\lambda_0}{2} + \boldsymbol{\mu}^T D_\mu^{-1} \mathbf{1} \frac{\lambda_1}{2}
\end{aligned}
$$

$$
\begin{aligned}
(D_\mu \mathbf{1})^T \check{\boldsymbol{\rho}} &= (D_\mu \mathbf{1})^T D_Q^{-1} \mathbf{1} V + (D_\mu \mathbf{1})^T D_Q^{-1} \mathbf{1} \frac{\lambda_0}{2} + (D_\mu \mathbf{1})^T D_Q^{-1} D_Q^{-1} \mathbf{1} \frac{\lambda_1}{2} \\
(D_\mu \mathbf{1})^T \check{\boldsymbol{\rho}} - (D_\mu \mathbf{1})^T D_Q^{-1} \mathbf{1} V &= (D_\mu \mathbf{1})^T D_Q^{-1} \mathbf{1} \frac{\lambda_0}{2} + (D_\mu \mathbf{1})^T D_Q^{-1} D_Q^{-1} \mathbf{1} \frac{\lambda_1}{2} \\
1 - \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1} V &= \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1} \frac{\lambda_0}{2} + \boldsymbol{\mu}^T D_Q^{-1} D_Q^{-1} \mathbf{1} \frac{\lambda_1}{2}
\end{aligned}
$$

$$
\begin{aligned}
\begin{bmatrix} 1 & \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1} \\ \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1} & \boldsymbol{\mu}^T D_Q^{-1} D_Q^{-1} \mathbf{1} \end{bmatrix} \begin{bmatrix} \frac{\lambda_0}{2} \\ \frac{\lambda_1}{2} \end{bmatrix} &= \begin{bmatrix} 0 \\ 1 - \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1} V \end{bmatrix} \\
\begin{bmatrix} \frac{\lambda_0}{2} \\ \frac{\lambda_1}{2} \end{bmatrix} &= \begin{bmatrix} 1 & \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1} \\ \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1} & \boldsymbol{\mu}^T D_Q^{-1} D_Q^{-1} \mathbf{1} \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 1 - \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1} V \end{bmatrix} \\
\frac{\lambda_0}{2} &= \frac{(\boldsymbol{\mu}^T D_Q^{-1} \mathbf{1})^2 V - \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1}}{\boldsymbol{\mu}^T D_Q^{-1} D_Q^{-1} \mathbf{1} - (\boldsymbol{\mu}^T D_Q^{-1} \mathbf{1})^2} \\
\frac{\lambda_1}{2} &= \frac{1 - \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1} V}{\boldsymbol{\mu}^T D_Q^{-1} D_Q^{-1} \mathbf{1} - (\boldsymbol{\mu}^T D_Q^{-1} \mathbf{1})^2}
\end{aligned}
$$

$$
\check{\boldsymbol{\rho}} = D_Q^{-1} \mathbf{1} V + D_Q^{-1} \mathbf{1} \frac{(\boldsymbol{\mu}^T D_Q^{-1} \mathbf{1})^2 V - \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1}}{\boldsymbol{\mu}^T D_Q^{-1} D_Q^{-1} \mathbf{1} - (\boldsymbol{\mu}^T D_Q^{-1} \mathbf{1})^2} + D_Q^{-1} D_Q^{-1} \mathbf{1} \frac{1 - \boldsymbol{\mu}^T D_Q^{-1} \mathbf{1} V}{\boldsymbol{\mu}^T D_Q^{-1} D_Q^{-1} \mathbf{1} - (\boldsymbol{\mu}^T D_Q^{-1} \mathbf{1})^2}
$$

$$
\check{\rho}_a = \frac{V}{Q_a} + \frac{\frac{1}{Q_a}(\mathbb{E}_\mu[\frac{1}{Q}]^2 V - \mathbb{E}_\mu[\frac{1}{Q}]) + \frac{1}{Q_a^2}(1 - \mathbb{E}_\mu[\frac{1}{Q}] V)}{\mathbb{E}_\mu[\frac{1}{Q^2}] - \mathbb{E}_\mu[\frac{1}{Q}]^2}
$$

Appendix C Supplement for Chapter 5 (An Idiosyncrasy of Time-discretization)

This appendix contains supplementary information for the contributions of Chapter 4 pertaining to an idiosyncrasy of time-discretization.

C.1 Servo Reacher Environment Details

The environment state $\mathbf{x}$ is a column vector containing the DC motor's angular velocity [rad/s], the DC motor's current [A], the output shaft's angle [rad], the output shaft's angular velocity [rad/s], and the output shaft's target angle [rad], respectively. The state vector is updated as follows:

$$\dot{\mathbf{x}}_t \leftarrow \begin{bmatrix} -\frac{b_m}{J_m} & \frac{K_t}{J_m} & 0 & 0 & 0 \\ -\frac{K_t}{L_a} & -\frac{R_a}{L_a} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -\frac{b_m}{J_m N \eta} & \frac{K_t}{J_m N \eta} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x}_t + \begin{bmatrix} 0 \\ \frac{1}{L_a} \\ 0 \\ 0 \\ 0 \end{bmatrix} A_t$$

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \dot{\mathbf{x}}_t \Delta_s$$

where $\Delta_s = 10^{-4}$ [s] is the simulation discretization granularity, and A_t is an input voltage with built-in saturation limits of $\in [-12, 12]$ [V]. The output shaft angle is clamped $\in [-1.306, 1.306]$ [rad] in accordance with Mahmood et al. (2018b). The physical parameters used are detailed below:

L_a	Armature Inductance	2.05×10^{-3} [H]
R_a	Armature Resistance	8.29 [Ohm]
J_m	Rotor Inertia	8.67×10^{-8} [$\text{kg} \cdot \text{m}^2$]
b_m	Rotor Friction	8.87×10^{-8} [$\text{N} \cdot \text{m} \cdot \text{s}$]
K_t	Torque Constant	0.0107 [$\frac{\text{N}\cdot\text{m}}{\text{A}}$]
N	Gear Ratio	200
η	Gear Efficiency	0.836

Given a target discretization interval $> 10^{-4}$ [s], the above updates are repeated until the target elapsed time is reached, keeping track of any overshoot and compensating accordingly in the next time interval. As a reinforcement learning environment, an agent observes the output shaft's angle, angular velocity, and target angle. The initial output shaft angle, θ_0, and target angle, θ_{target}, are uniformly sampled $\in [-1.306, 1.306]$ at the start of each episode, and an episode terminates when $|\theta_{t+1} - \theta_{target}| < 0.1$ [rad] with angular velocity $\dot{\theta}_{t+1} < 0.1$ [rad/s]. An agent provides a continuous-valued action as a voltage, and receives a reward $|\theta_{t+1} - \theta_{target}|$, computed and received jointly with the next observation.