A Unified View of Multi-step Temporal Difference Learning

by

Kristopher De Asis

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

Master of Science

Department of Computing Science

University of Alberta

Abstract

Temporal-difference (TD) learning is an important approach for predictive knowledge representation and sequential decision making. Within TD learning exists multi-step methods which unify one-step TD learning and Monte Carlo methods in a way where intermediate algorithms can outperform either extreme. Multi-step TD methods allows a practitioner to address a bias-variance trade-off between reliance on current estimates, which could be poor, and incorporating longer sequences of sampled information, which could have large variance. In this dissertation, we investigate an extension of multi-step TD learning aimed at reducing the variance in the estimates, and provide a unified view of the space of multi-step TD algorithms.

In Monte Carlo methods, information about the error of a known quantity is sometimes incorporated in an attempt to reduce the error in the estimation of an unknown quantity. This is known as the method of control variates, and has not been extensively explored in TD learning. We show that control variates can be formulated in multi-step TD learning, and demonstrate their improvement in terms of learning speed and accuracy. We then show how the inclusion of control variates provides a deeper understanding of how n-step TD methods relate to TD(λ) algorithms.

We then look at a previously proposed method to unify the space of n-step TD algorithms, the n-step Q(σ) algorithm. We provide empirical results and analyze properties of this algorithm, suggest an improvement based on insight from the control variates, and derive the TD(λ) version of the algorithm. This

generalization can recover existing multi-step TD algorithms as special cases, providing an alternative, unified view of them.

Lastly, we bring attention to the discount rate in TD learning. The discount rate is typically used to specify the horizon of interest in sequential decision making problems, but we introduce an alternate view of the parameter with insight from digital signal processing. By allowing the discount rate to take on complex numbers within the complex unit circle, we can extend the types of knowledge learnable by a TD agent into the frequency domain. This allows for online and incremental estimation of the extent at which particular frequencies exist in a signal, with the standard discounting framework corresponding to the zero frequency case.

Preface

Several chapters in this thesis have content based on published papers, or papers in progress. One of these papers had equal contribution with two co-authors. I have outlined all relevant portions of Chapters 3, 4, and 5 below. The remaining contributions, particularly the latter two sections of Chapter 4, were originally produced for this thesis.

Chapter 3 of this thesis is based on a published paper (De Asis and Sutton, 2018) which resulted from an approach to reduce the variance in multi-step TD methods. I was responsible for exploring the connection between n-step TD methods with control variates and their TD(λ) counterparts, as well as the view as an alternative way to extend Expected Sarsa to the multi-step setting. The idea to focus on the explicit introduction of the control variate terms was motivated by personal communication with Mahmood.

The first three sections of Chapter 4 are based on a published paper (De Asis, Hernandez-Garcia, Holland, and Sutton, 2018) which aimed to analyze and provide first empirical results on the Q(σ) algorithm. The Q(σ) algorithm was originally proposed by Precup, Sutton, and Singh (2000). Holland was responsible for the results in the stochastic windy gridworld environment of Section 4.3.2, and Hernandez-Garcia was responsible for the results in the mountain cliff environment of Section 4.3.3. I was responsible for the results in the 19-state random walk, the analysis behind why intermediate values of σ might perform either extreme, and the simple heuristic method for dynamically varying the σ parameter.

Chapter 5 of this thesis is based on a paper under review (De Asis et al., 2018). In this work, I discovered that meaningful information can be learned when the discount rate parameter is allowed to take on complex numbers. I

made the connection between complex returns and the Discrete Fourier Transform of digital signal processing literature, and motivated the idea of a TD agent being able to identify periodic structure in the return.

Acknowledgements

I would like to first thank my supervisor, Richard Sutton, for his guidance throughout my graduate studies. I am fortunate to have had many open ended discussions with him regarding our views on artificial intelligence, and how our minds work. Our conversations have resulted in an emphasis to not only make my ideas clear to myself, but to find ways to clearly deliver them to others.

I am grateful to my co-authors on my first paper. Working closely with J. Fernando Hernandez-Garcia and G. Zacharias Holland helped to develop ideas quicker, maintain the quality of the work done, and provide a good foundation on how to approach future research ideas.

I am thankful for the entire Reinforcement Learning & Artificial Intelligence group at the University of Alberta, as they are a supportive, friendly, and dynamic group with many shared interests and values. My discussions within the group have led to the development of a more well-rounded set of intuitions and perspectives on artificial intellgence-related topics.

I would like to acknowledge Alberta Innovates Technology Futures, Google Deepmind, the Natural Sciences and Engineering Research Council of Canada, the Alberta Machine Intelligence Institute, and the University of Alberta for funding my work throughout my graduate studies.

Contents

List of Figures

Chapter 1

Introduction

The ability to learn online and incrementally from raw sensorimotor experience provides a powerful and scalable framework for an intelligent agent. It would allow for adaptation in situations that haven't been considered, and not have to wait until the final outcome before making adjustments. This idea of behaving in an environment and learning from experience is well captured in a reinforcement learning setting, where an agent takes actions in an environment, and receives feedback signals from which it can learn about.

Temporal Difference (TD) methods are an important approach for reinforcement learning problems as they allow an agent to learn long-term predictions of signals of interest. Characteristic to TD is that it incrementally adjusts its estimates based on the outcome of each decision. For example, when playing a video game, an agent would not have to wait until the game ends to update its predictions about what happens later in the game. This is accomplished through *bootstrapping*, where an agent learns a prediction based on a combination of observed outcomes and its current guess of what will happen next. Such predictions form a powerful knowledge representation that an agent can continually test and update, and potentially base its decisions on.

Multi-step TD methods create a spectrum of algorithms which unify one-step TD learning and Monte Carlo methods in a way where intermediate algorithms can outperform either extreme. One-step TD methods only use information from a single time step when computing an estimate, resulting in large bias as it relies on its estimate of the next time step onwards to be accu-

rate. Monte Carlo methods include information from all time steps, resulting in large variance due to the variability in longer sampled sequences. Multi-step TD methods allow for use of sampled information across multiple time steps before bootstrapping off of its current estimates, addressing this bias-variance tradeoff between the two extremes.

Off-policy learning in reinforcement learning involves an agent learning long-term predictions given that it had behaved differently from the behavior which generated its experience. It can be interpreted as an agent answering counterfactual questions about its experience, and allows the agent to learn many predictions simultaneously, vastly expanding the world knowledge it can gather from a single stream of experience. Off-policy learning is a more general setting, as on-policy learning can be viewed as the case where an agent is learning about its current behavior. TD algorithms have several extensions to the off-policy setting but, especially for multi-step TD methods, they often suffer from having large variance.

In this dissertation, we explore a technique for reducing the variance in the predictions learned by off-policy multi-step TD methods. We then show that this extension provides a deeper understanding of how existing methods relate to one another. Using this insight, we build upon a previously proposed algorithm to unify the space of multi-step TD methods, and contextualize recent multi-step methods within the resulting algorithm. Lastly, we extend the types of predictive knowledge learnable by TD methods into the frequency domain, providing another direction to explore in the space of algorithms.

1.1 Related Work

Multi-step TD methods were first extended to the off-policy setting by Precup, Sutton, and Singh (2000). One method made use of per-decision importance sampling, but it was readily apparent that the use of importance sampling ratios introduced large variance in the estimates. Several proposals have been made to address this variance issue in the TD(λ) space of multi-step TD methods (Precup et al., 2006; Mahmood et al., 2014; Mahmood and Sutton, 2015;

Mahmood et al., 2015; Hallak et al., 2015; Munos et al., 2016; Mahmood et al., 2017; Yu et al., 2017). Work has also been done with in offline approaches to sequential decision making, such as the use of doubly robust estimation (Jiang et al., 2016; Thomas et al., 2016). Some work has been done in the n-step TD space of multi-step TD methods (Sutton et al., 2014; van Seijen et al., 2014; Mahmood and Sutton, 2015), but n-step TD methods have gotten relatively less attention compared to TD(λ) methods. How such modifications in the TD(λ) space of algorithms affects their n-step TD counterparts has not been thoroughly explored.

The idea of value functions representing predictive knowledge of the environment can be traced back to Sutton (1988). The idea was later formalized in a *general value function* (GVF) framework by Sutton et al. (2011), where value functions are given an interpretation as an answer to a predictive question. However, whether meaningful questions can be posed by relaxing the range of values the discount rate can take on hasn't been thoroughly explored.

1.2 Contributions

This dissertation claims the following contributions towards providing a unified view of multi-step TD learning:

1. **n-step Control Variates**: Our first contribution is the formulation of control variates for n-step TD learning algorithms. This results in the n-step SCV Sarsa and n-step ACV Sarsa algorithms. We demonstrate their improvement over comparable n-step TD methods, provide intuitive insight for the introduced control variates, and show how their implications in the TD(λ) class of multi-step algorithms.

2. **The n-step Q(σ) Algorithm**: Next, we provide analysis and empirical results for an original proposal to unify the space of n-step methods. We further extend the algorithm with the aforementioned n-step control variates, resulting in the n-step CV Q(σ) algorithm.

3. **The Q(σ, λ) Algorithm**: We then derive the TD(λ) version of the n-step CV Q(σ) algorithm. We show that this algorithm can recover existing TD(λ) methods, and provides an alternate view of action-dependent eligibility traces.

4. **Complex Value Functions**: Our final contribution is an extension to the predictive knowledge representable by value functions. By relaxing the typical range of values the discount rate can be set to, it can be shown that a TD learning agent can incrementally learn the expected Discrete Fourier Transform of the discounted return.

Chapter 2

Background on Reinforcement Learning

This chapter aims to provide background information relevant to the contributions claimed by this thesis. In particular, it introduces the agent-environment interaction model of reinforcement learning, defines the predictive knowledge-based approach to reinforcement learning tasks, and explains how TD methods address it.

2.1 The Reinforcement Learning Setting

Central to all reinforcement learning problems is the agent-environment interaction model, as depicted by Figure 2.1. In this model, it is assumed that an agent exists in some *state* of the environment, whether it's a distinct identifier or a set of features which characterize it. The agent interacts with the environment by selecting an available *action*, which results in the environment producing the agent's next state along with a *reward*. This interaction model is typically formulated as or built on top of a Markov Decision Process (MDP).

Under the MDP framework, an agent interacts with an environment over a sequence of discrete time steps. At each time step t, the agent receives information about the environment's current state, $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states in the MDP. The agent then uses this state information to select an action, $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(s)$ is the set of possible actions in state s. Based on the environment's current state and the agent's selected

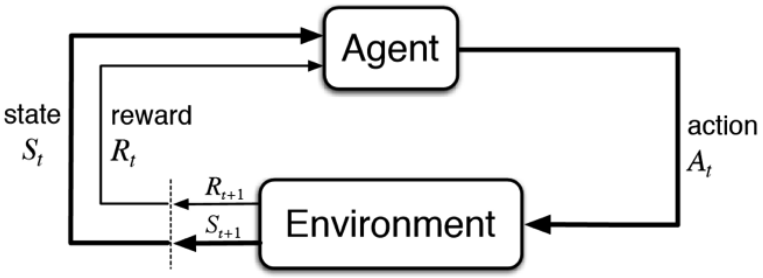

Figure 2.1: The agent-environment interaction model. Reprinted from Sutton and Barto, 2018.

action, the agent receives a reward, $R_{t+1} \in \mathbb{R}$, and gets information about the environment's next state, $S_{t+1} \in \mathcal{S}$, according to the *environment model*: $p(r, s'|s, a) \stackrel{\text{def}}{=} Pr(R_{t+1} = r, S_{t+1} = s'|S_t = s, A_t = a)$.

The agent selects its actions according to a *policy*, defined to be a probability distribution across actions given a state: $\pi(s, a) \stackrel{\text{def}}{=} Pr(A_t = a|S_t = a)$. Behaving under this policy, the agent is interested in a *return*, defined to be a cumulative discounted sequence of rewards:

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

given a discount rate $\gamma \in [0, 1]$ and T equal to the final time step in an episodic setting, or $\gamma \in [0, 1)$ and T equal to infinity for a continuing setting. In a reinforcement learning control task, where the rewards specify some desired outcome, the agent's goal to learn the policy where behaving under it will maximize the expected return.

2.2 Value-based Methods

Value-based methods are a predictive-knowledge approach for the reinforcement learning setting, and are characterized by computing *value functions*. The two common types of value functions are the *state-value function*, defined to be the expected return when starting in state s and following policy π:

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t|S_t = s] \tag{2.2}$$

and the *action-value function*, defined to be the expected return when starting in state s, selecting action a, and following policy π afterwards:

$$q_\pi(s,a) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{2.3}$$

Of note, the action-value function provides more information, and the state-values can be computed as the expected action-value under the policy π for a given state:

$$v_\pi(s) = \sum_a \pi(s,a) q_\pi(s,a) \tag{2.4}$$

They address the *policy evaluation* problem in reinforcement learning, where the goal is to accurately predict what the return will be for a given policy. That said, they can also be used to address control problems, as the predicted returns can be used to compare actions and derive a new policy. In particular, acting greedily with respect to the action-values under some policy will result in a strictly improved policy. This notion of *policy improvement* requires that for all state-action pairs, the values under the new policy will be greater than or equal to the values under the previous policy (Sutton and Barto, 2018). Alternating the process of policy evaluation and policy improvement is known as *policy iteration*, and will converge to an optimal policy which maximizes the expected return.

2.3 Offline Algorithms

In this section we detail two offline approaches to the policy evaluation problem which motivate the ideas behind TD learning.

2.3.1 Dynamic Programming

The definition for the state-value function in Equation 2.2 can be rewritten in terms of the values of successor states as follows:

$$v_\pi(s) = \sum_a \pi(s,a) \sum_{s',r} p(s',r|s,a)\big(r + \gamma v_\pi(s')\big) \tag{2.5}$$

This is referred to as the *Bellman equation* for v_π, where v_π is the fixed point solution that satisfies the recursion relationship. With an approximate value

function $V \approx v_\pi$, we can evaluate the Bellman equation directly. Doing so involves *bootstrapping* off of the current estimated values of successor states s' to update the value of each state s:

$$\forall s : V(s) \leftarrow \sum_a \pi(s,a) \sum_{s',r} p(s',r|s,a)\big(r + \gamma V(s')\big) \tag{2.6}$$

Repeatedly updating the value of each state until convergence results in the *dynamic programming* approach to policy evaluation (Bellman, 1957; Howard, 1960). Similarly, the Bellman equation for q_π and corresponding dynamic programming update with approximate value function $Q \approx q_\pi$ are:

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\big(r + \gamma \sum_{a'} \pi(s',a') q_\pi(s',a')\big) \tag{2.7}$$

$$\forall s,a : Q(s,a) \leftarrow \sum_{s',r} p(s',r|s,a)\big(r + \gamma \sum_{a'} \pi(s',a') Q(s',a')\big) \tag{2.8}$$

A key idea behind the dynamic programming approach is that a better estimate for a value can be computed based on its current estimates for other values. However, it requires explicit knowledge of the environment model $p(s',r|s,a)$ to evaluate the Bellman equation.

2.3.2 Monte Carlo Methods

Another approach to approximating the value function is to perform a Monte Carlo estimate of the expected return. In the episodic setting, an agent could perform several rollouts under a policy π until termination, storing the sequence of state-action pairs and rewards. For each visited state, the return G_t from that point onward can be computed and averaged together with other returns sampled from the same state. Using an exponential moving average with $\alpha \in (0,1]$, we get the following update for each time step in the trajectory:

$$V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha G_t \tag{2.9}$$

Of note, the above average can be rewritten as taking a step proportional to the difference between the sample and the current estimate with a step size α:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \tag{2.10}$$

The above update is for the case where the rollouts are performed under the policy we are trying to evaluate. In the off-policy case, we aim to estimate the values under a *target policy* while data is being generated from a potentially different *behavior policy*. One technique for correcting this discrepancy is the use of *importance sampling* weights (Hammersley and Handscomb 1964; Rubinstein 1981). With a target policy π and behavior policy μ, the importance sampling weight of a single transition is defined to be:

$$\rho_t \stackrel{\text{def}}{=} \frac{\pi(S_t, A_t)}{\mu(S_t, A_t)} \tag{2.11}$$

The importance sampling weight for a trajectory can then be computed as a product of importance sampling ratios:

$$\prod_{k=t}^{T-1} \frac{\pi(S_k, A_k)p(S_{k+1}|S_k, A_k)}{\mu(S_k, A_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(S_k, A_k)}{\mu(S_k, A_k)} = \prod_{k=t}^{T-1} \rho_k \tag{2.12}$$

Assuming full coverage, that is, all of the actions that may occur under the target policy may also occur under the behavior policy, the expected value of the above product of importance sampling ratios is 1. This can be shown by summing over all possible trajectories $\tau = S_t, A_t, S_{t+1}, A_{t+2}, ...S_T$:

$$\begin{aligned}
\mathbb{E}_\mu\left[\prod_{k=t}^{T-1} \rho_k \middle| S_t = s\right] &= \sum_\tau \left(\prod_{k=t}^{T-1} \mu(S_k, A_k)p(S_{k+1}|S_k, A_k)\right)\left(\prod_{k=t}^{T-1} \frac{\pi(S_k, A_k)}{\mu(S_k, A_k)}\right) \\
&= \sum_\tau \prod_{k=t}^{T-1} \pi(S_k, A_k)p(S_{k+1}|S_k, A_k) \\
&= 1
\end{aligned}$$

Weighting a trajectory sampled by the behavior policy's distribution by the product of importance sampling ratios will result in computing the expectation under the target policy's distribution (also assuming full coverage). This can be seen if we denote some trajectory-dependent random variable X

and compute the expectation for the state at time t:

$$\begin{aligned}\mathbb{E}_\mu\left[\left(\prod_{k=t}^{T-1}\rho_k\right)X_t\middle|S_t=s\right] &= \sum_{x,\tau} x\left(\prod_{k=t}^{T-1}\frac{\pi(S_k,A_k)}{\mu(S_k,A_k)}\right)\left(\prod_{k=t}^{T-1}\mu(S_k,A_k)p(S_{k+1}|S_k,A_k)\right)\\ &= \sum_{x,\tau} x\left(\prod_{k=t}^{T-1}\pi(S_k,A_k)p(S_{k+1}|S_k,A_k)\right)\\ &= \mathbb{E}_\pi\left[X_t\middle|S_t=s\right]\end{aligned}$$

There are several ways in which importance sampling can be used to correct the discrepancy between the behavior and target policy distributions (Sutton et al., 2014; Mahmood and Sutton, 2015). For example, it could be used to weight the return, or the differences between the return and the estimates:

$$V(S_t) \leftarrow V(S_t) + \alpha\left(\left(\prod_{k=t}^{T-1}\rho_k\right)G_t - V(S_t)\right) \tag{2.13}$$

$$V(S_t) \leftarrow V(S_t) + \alpha\left(\prod_{k=t}^{T-1}\rho_k\right)(G_t - V(S_t)) \tag{2.14}$$

A drawback of the importance sampling technique is that the importance sampling weights can become arbitrarily large, resulting in large variance. Between the above two methods, the latter is typically preferred as the difference between the return and the estimate tends to be a smaller and less variable number than the return alone (Jiang et al., 2016). Another approach is to only apply the importance sampling weight to the terms affected by each action selected. This is referred to as *per-decision importance sampling* (Precup et al., 2000), and estimates the return as follows:

$$G_t = \rho_{t+1}R_{t+1} + \gamma\rho_{t+1}\rho_{t+2}R_{t+2} + \gamma^2\rho_{t+1}\rho_{t+2}\rho_{t+3}R_{t+3} + ... \tag{2.15}$$

Per-decision importance sampling has the appeal of computing a smaller product of importance sampling ratios for earlier terms in the trajectory, potentially mitigating unnecessary variance. However, it lacks the intuitive benefit of scaling a difference as in Equation 2.14. The remainder of this thesis will focus on this per-decision setting.

An appealing aspect of Monte Carlo methods are that they don't require a model of the environment, and can learn entirely from experience. However,

Monte Carlo methods are restricted to episodic settings as the complete return needs to be sampled for learning to occur.

2.4 One-step TD Learning

Combining the model-free sampling techniques of Monte Carlo learning with the bootstrapping idea of dynamic programming results in the TD learning approach to policy evaluation. If we recall the Bellman equation for v_π in Equation 2.5, we can compute a Monte Carlo sample of the recursive relationship. Given a transition $\{S_t, A_t, R_{t+1}, S_{t+1}\}$ sampled from the environment model $p(s', r|s, a)$ and the target policy π, we can compute the following estimate of the return with an approximate value function V:

$$\hat{G}_t = R_{t+1} + \gamma V(S_{t+1}) \tag{2.16}$$

This estimate is sometimes denoted as the *TD target*. After computing this estimate, the value of the previous state can be updated with a step proportional to the difference between the TD target and the current value estimate, similar to Equation 2.10:

$$V(S_t) \leftarrow V(S_t) + \alpha(\hat{G}_t - V(S_t)) \tag{2.17}$$

where the term in brackets in Equation 2.17 is often referred to as the *TD error*. This allows for online and incremental learning, as these updates can be performed on a per-decision basis without waiting for the end result. Extending this to the off-policy setting with per-decision importance sampling, the TD target in Equation 2.16 becomes:

$$\hat{G}_t = \rho_t(R_{t+1} + \gamma V(S_{t+1})) \tag{2.18}$$

In the action value case, based on the Bellman equation for q_π in Equation 2.7, we can perform the following update with knowledge of the next sampled action A_{t+1}:

$$\hat{G}_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) \tag{2.19}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(\hat{G}_t - Q(S_t, A_t)) \tag{2.20}$$

This results in the *Sarsa* algorithm, often presented as the classical on-policy algorithm (Rummery and Niranjan, 1994; Sutton, 1996) for action-values. Of note, the immediate action is specified for action-values, and thus the immediate reward does not depend on the policy. Due to this, in the off-policy setting, the importance sampling correction only needs to be applied on the term after the immediate reward in Equation 2.19:

$$\hat{G}_t = R_{t+1} + \gamma \rho_{t+1} Q(S_{t+1}, A_{t+1}) \tag{2.21}$$

Another approach to the action-value case is to compute the expectation across action-values under the target policy in the next state directly. Denoting the expected action-value under the target policy for a given state as $\bar{Q}_t$, we get the following TD target:

$$\bar{Q}_t = \sum_a \pi(S_t, a) Q(S_t, a) \tag{2.22}$$

$$\hat{G}_t = R_{t+1} + \gamma \bar{Q}_{t+1} \tag{2.23}$$

Doing so results in the *Expected Sarsa* algorithm (van Seijen et al., 2009). It avoids unnecessary variance due to policy stochasticity by not relying on sampling the next action, and is being able to perform off-policy learning without importance sampling ratios. In control problems where the goal is to maximize the expected return, the target policy is typically set to one that is deterministically greedy with respect to the action-values. This is equivalent to performing a maximum operation on the action-values in the next state:

$$\hat{G}_t = R_{t+1} + \gamma \max_a Q(S_{t+1}, a) \tag{2.24}$$

The above TD target results in the *Q-learning* algorithm, a popular off-policy control method (Watkins, 1989). However, since it can be viewed as a special case of Expected Sarsa, we will focus on the general case with arbitrary target policies.

2.5 n-step TD Learning

TD algorithms are referred to as one-step TD algorithms when they only incorporate information from a single time step in the estimate of the return.

In multi-step TD methods, a longer sequence of experienced rewards is used to compute said estimate. For example, on-policy n-step TD (Sutton and Barto, 2018) would update a value $V(S_t)$ towards the following estimate:

$$\begin{aligned}\hat{G}_{t:t+n} &= R_{t+1} + \gamma R_{t+2} + ... + \gamma^n V(S_{t+n}) \\ &= \gamma^n V(S_{t+n}) + \sum_{k=0}^{n-1} \gamma^k R_{t+k+1}\end{aligned} \tag{2.25}$$

Multi-step methods create a spectrum of algorithms which unify one-step TD learning and Monte Carlo methods, and address a bias-variance trade-off (Jaakkola et al., 1994). One-step TD has lower variance as it only has to deal with the variability in the immediate reward, as opposed to that of a longer reward sequence. However, it has larger bias as it puts relatively more importance on bootstrapping off of its current estimates, which could be poor.

In the off-policy case with per-decision importance sampling, it's more convenient to express the estimate of the return recursively. First, we define the bootstrapping condition:

$$\hat{G}_{t:t} = V(S_t) \tag{2.26}$$

The return estimate for n-step TD then becomes:

$$\hat{G}_{t:t+n} = \rho_t(R_{t+1} + \gamma \hat{G}_{t+1:t+n}) \tag{2.27}$$

In the action-value setting, the return estimate for n-step Sarsa with per-decision importance sampling is:

$$\hat{G}_{t:t} = Q(S_t, A_t) \tag{2.28}$$

$$\hat{G}_{t:t+n} = R_{t+1} + \gamma \rho_{t+1} \hat{G}_{t+1:t+n} \tag{2.29}$$

For the Expected Sarsa algorithm, there have been several proposed extensions to the multi-step setting. One is the n-step Expected Sarsa algorithm (Sutton and Barto, 2018) which samples a sequence of rewards like n-step Sarsa, but bootstraps off of the expectation across action-values on the n-th step. This results in importance samping ratios for the reward sequence but not the final

step. It can be expressed with the same return estimate in Equation 2.29, but with the following bootstrapping condition:

$$\hat{G}_{t:t} = \frac{\bar{Q}_{t+1}}{\rho_t} \tag{2.30}$$

where the division by ρ_t is to cancel out the final importance sampling ratio in the recursive definition. Another extension of Expected Sarsa to the multi-step setting is the *n-step Tree-backup* algorithm (Precup et al., 2000), which is characterized by computing an expectation at every time step. Maintaining the bootstrapping condition of Equation 2.28, n-step Tree-backup's estimate of the return is:

$$\hat{G}_{t:t+n} = R_{t+1} + \gamma \left(\pi(S_{t+1}, A_{t+1}) \hat{G}_{t+1:t+n} + \sum_{a \neq A_{t+1}} \pi(S_{t+1}, a) Q(S_{t+1}, a) \right) \tag{2.31}$$

An appealing aspect of Tree-backup is that it can perform off-policy learning without importance sampling, like one-step Expected Sarsa. This results in lower variance, but introduces additional bias relative to n-step Expected Sarsa for the same n (De Asis et al., 2018). The additional bias is due to the $\pi(S_{t+1}, A_{t+1}) \hat{G}_{t+1:t+n}$ term in Equation 2.31. It can be seen as decaying the remainder of the sampled return based on the stochasticity of the target policy, effectively giving less weight to the reward sequence, and compensating by bootstrapping off of the action-values of actions not taken.

2.6 TD(λ) Methods

Another family of multi-step TD algorithms are the TD(λ) methods (Sutton and Barto, 2018). They are characterized by computing a geometrically weighted sum of n-step returns, denoted as the λ-return:

$$\hat{G}_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \hat{G}_{t:t+n} \tag{2.32}$$

It introduces a hyperparameter $\lambda \in [0, 1]$ where $\lambda = 0$ gives one-step TD learning, and increasing λ effectively increases the number of sampled rewards in the estimate of the return.

Substituting the return of on-policy n-step TD from Equation 2.25 into Equation 2.32 gives the λ-return for on-policy TD(λ):

$$\hat{G}_t^\lambda = V(S_t) + \sum_{k=t}^{\infty} \left(R_{k+1} + \gamma V(S_{k+1}) - V(S_k)\right) \prod_{i=t+1}^{k} \gamma\lambda$$

This shows that the λ-return for on-policy TD(λ) can be estimated by computing one-step TD errors, and decaying the weight of later TD errors at a rate of $\gamma\lambda$. Implementing this online and incrementally, an *eligibility trace* vector is maintained to track which states led to the current step's TD error. The traces of earlier states are decayed at each step by the afformentioned decay rate, and each state's value is adjusted by the current TD error weighted by the state's corresponding trace.

The above derivation is easier to show via an equivalent recursive definition for the λ-return (Assuming the value function does not change):

$$\begin{aligned}
\hat{G}_t^\lambda &= R_{t+1} + \gamma\big((1-\lambda)V(S_{t+1}) + \lambda\hat{G}_{t+1}^\lambda\big) \\
\hat{G}_t^\lambda &= R_{t+1} + \gamma V(S_{t+1}) + \gamma\lambda(\hat{G}_{t+1}^\lambda - V(S_{t+1})) \\
\hat{G}_t^\lambda - V(S_t) &= R_{t+1} + \gamma V(S_{t+1}) - V(S_t) + \gamma\lambda(\hat{G}_{t+1}^\lambda - V(S_{t+1})) \\
\hat{G}_t^\lambda - V(S_t) &= \sum_{k=t}^{\infty} \left(R_{k+1} + \gamma V(S_{k+1}) - V(S_k)\right) \prod_{i=t+1}^{k} \gamma\lambda \\
\hat{G}_t^\lambda &= V(S_t) + \sum_{k=t}^{\infty} \left(R_{k+1} + \gamma V(S_{k+1}) - V(S_k)\right) \prod_{i=t+1}^{k} \gamma\lambda
\end{aligned}$$

Due to how traces are updated after each decision, TD(λ) methods lend themselves nicely to certain per-decision importance sampling methods. For example, if we substitute the return of n-step Sarsa with per-decision importance sampling from Equation 2.29 into Equation 2.32, we get the following λ-return:

$$\hat{G}_t^\lambda = R_{t+1} + \gamma((1-\lambda)\rho_{t+1}Q(S_{t+1}, A_{t+1}) + \lambda\rho_{t+1}G_{t+1}^\lambda) \tag{2.33}$$

Solving this recursion, again assuming that the value estimates do not change, results in:

$$\hat{G}_t^\lambda = V(S_t) + \sum_{k=t}^{\infty} \left(R_{k+1} + \gamma\rho_{k+1}Q(S_{k+1}, A_{k+1}) - Q(S_k, A_k)\right) \prod_{i=t+1}^{k} \gamma\lambda\rho_i \tag{2.34}$$

Updating towards this estimate of the return results in the Sarsa(λ) algorithm, which can similarly be estimated as a sum one-step Sarsa's TD errors, weighted by eligibility traces with a trace decay rate of $\gamma\lambda\rho_i$.

Similarly, substituting the return of n-step Tree-backup from Equation 2.31 results in:

$$\hat{G}_t^{\lambda} = V(S_t) + \sum_{k=t}^{\infty} \left(R_{k+1} + \gamma\bar{Q}_{t+1} - Q(S_k, A_k)\right) \prod_{i=t+1}^{k} \gamma\lambda\pi(S_i, A_i) \tag{2.35}$$

This shows that the λ-return of the Tree-backup(λ) algorithm can be estimated as a sum of one-step Expected Sarsa's TD errors, with a trace decay rate of $\gamma\lambda\pi(S_i, A_i)$. Of note, $\pi(S_i, A_i) \leq \rho_i$, resulting in traces which decay quicker than those of the Sarsa(λ) algorithm. This captures the afformentioned additional bias of the Tree-backup algorithm, as it effectively updates fewer values of previous state-action pairs for a given value of λ.

n-step Expected Sarsa does not have a λ-return which can be expressed as a sum of TD errors of existing one-step TD methods. This is due to how it performs a mix of sampling and computing expectations, and does not commit to one at each time-step. For example, on-policy 2-step Expected Sarsa would sample at time steps $t+1$ and $t+2$, and compute an expectation across actions in the state at step $t+3$:

$$\hat{G}_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 \bar{Q}_{t+2}$$

Incrementing the time steps allows us to see what is computed at the next step:

$$\hat{G}_{t+1} = R_{t+2} + \gamma R_{t+3} + \gamma^2 \bar{Q}_{t+3}$$

From this, we can see that the expectation computed at step $t+2$ has become a sampled reward, and a new expectation is computed at step $t+3$. TD(λ) methods commit to a sample or expectation at a given step through the one-step TD error used, and the amount the traces are decayed by.

Because TD(λ) methods aim to estimate a multi-step return with a sum of one-step TD errors, the action-value setting involves subtracting an action-value of a specific state-action pair. This subtraction can be viewed as canceling out a bootstrapping term of a previous n-step return, and replacing it

with the reward and value of the next state. Since the bootstrapping conditions in the recursive definitions of n-step Sarsa and n-step Tree-backup are an individual action-value, there is a clear notion of what gets canceled out with the subtraction and what gets replaced with the information of the next step. This is not the case with n-step Expected Sarsa, as seen from its bootstrapping condition in Equation 2.30.

Contrasting with n-step TD methods, the computational complexity of TD(λ) control algorithms scales with the size of the environment, $|\mathcal{S}| \times |\mathcal{A}|$. That is, there is an environment-specific increase in complexity, but it no longer scales with the number of sampled rewards in the estimate of the return. A notable advantage of n-step methods is the conceptual clarity, as well as exact computation of the multi-step returns, since the incremental estimation of the λ-return assumes the value function isn't changing.

2.7 Extension to Function Approximation

The aforementioned algorithms have been presented as tabular solution methods, where every state is fully observed. Many applications involve impractically large state spaces, or partial observability of the underlying state. Due to this, function approximation is used. The use of function approximation allows for a more compact representation of the value function, as well as potential generalization between similar states in terms of observed features.

This is accomplished by parametrizing the value function with a set of weights, which we will denote with w. A natural extension to tabular solution methods is the use of linear function approximation, as a tabular representation can be viewed as a linear function where the state features consist of indicator vectors. However, any arbitrary function can be used, such as a non-linear one.

The goal is then to compute a set of weights which make the approximate value function as close to the true value function as possible. This is done by defining an optimization objective, and searching for the best set of weights under this objective. One possible objective is the *mean-squared value error*

(MSVE). For a parametrized state-value function $\hat{v}(s, w)$, and a *behavior distribution* $d_\mu(s)$ capturing the fraction of the time spent in a state s under the behavior policy:

$$\text{MSVE}(w) \stackrel{\text{def}}{=} \frac{1}{2} \sum_{s \in \mathcal{S}} d_\mu(s)(v_\pi(s) - \hat{v}(s, w))^2 \tag{2.36}$$

One way to optimize this objective is to use *stochastic gradient descent* (Widrow and Hoff, 1960), where a sample of the gradient of the objective is taken, and the weights are updated in the direction of the sample with step size α. Treating w as a column vector of weights:

$$w_{t+1} \leftarrow w_t + \frac{1}{2}\alpha \nabla_w (v_\pi(s) - \hat{v}(s, w_t))^2 \tag{2.37}$$

Where $\nabla_w f(w)$ is the gradient of a function $f(w)$ with respect to the parameters w. For a function with d parameters:

$$\nabla_w f(w) \stackrel{\text{def}}{=} \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, ..., \frac{\partial f(w)}{\partial w_d} \right)^T \tag{2.38}$$

Since TD methods don't have access to the true value function $v_\pi(s)$, they would use the estimate of the return $\hat{G}_t$ in its place. Since the bootstrapped values depend on the current set of weights, we end up with a biased estimate of the gradient, and end up performing *semi-gradient* updates as it's no longer a true gradient descent method (Sutton, 1988). For example, one-step semi-gradient TD would perform the following update:

$$\begin{aligned} \hat{G}_t &\leftarrow R_{t+1} + \gamma \hat{v}(S_{t+1}, w_t) \\ w_{t+1} &\leftarrow w_t + \alpha(\hat{G}_t - \hat{v}(S_t, w_t)) \nabla_w \hat{v}(S_t, w_t) \end{aligned} \tag{2.39}$$

To give a concrete example, let's consider the case of linear function approximation. For a particular state, an agent observes a k-dimensional feature vector which we denote as $\phi(S_t) \in \mathbb{R}^k$. Through a k-dimensional weight vector $w_t \in \mathbb{R}^k$, the value function is represented as a linear combination of these features:

$$\hat{v}(S_t, w_t) = w_t^T \phi(S_t) \tag{2.40}$$

Under linear function approximation, the gradient of the MSVE objective with respect to the weight vector is then:

$$\nabla_w \hat{v}(S_t, w_t) = \phi(S_t) \tag{2.41}$$

resulting in the following update:

$$\begin{aligned} \hat{G}_t &\leftarrow R_{t+1} + \gamma w_t^T \phi(S_{t+1}) \\ w_{t+1} &\leftarrow w_t + \alpha(\hat{G}_t - w_t^T \phi(S_t))\phi(S_t) \end{aligned} \tag{2.42}$$

The tabular setting can be viewed as a special case of linear function approximation where each state has a unique indicator vector. For notational clarity, we will use a tabular notation for the work presented in this thesis.

Although other optimization objectives exist, as well as TD methods which perform true gradient updates (Sutton et al., 2009), we will focus on semi-gradient methods with the MSVE objective in this thesis whenever we mention the use of function approximation.

Chapter 3

Per-decision Control Variates

This chapter describes the idea of control variates and its application in Monte Carlo estimation. This is done to investigate whether control variates can address the large variance of multi-step TD methods mentioned earlier in this thesis. The chapter then derives per-decision control variates for multi-step TD learning, provides intuition on the potential variance reduction, and shows how the introduction of these control variates provides a deeper understanding of existing TD methods. Lastly, it provides empirical results which demonstrate the benefit of using control variates in multi-step TD learning.

3.1 The Control Variate Method

When trying to estimate the expectation of some variable X, control variates are often of the following form (Ross, 2013):

$$X^* = X + c(Y - \mathbb{E}[Y]) \tag{3.1}$$

where Y is the outcome of another variable with a known expected value, and c is a coefficient to be set. X^* then has the following variance:

$$\text{Var}(X^*) = \text{Var}(X) + c^2\text{Var}(Y) + 2c\text{Cov}(X, Y) \tag{3.2}$$

This variance can be minimized with the optimal coefficient c^*:

$$\begin{aligned}\frac{\partial \text{Var}(X^*)}{\partial c} &= 2c\text{Var}(Y) + 2\text{Cov}(X, Y) \\ 0 &= 2c^*\text{Var}(Y) + 2\text{Cov}(X, Y) \\ c^* &= -\frac{\text{Cov}(X, Y)}{\text{Var}(Y)} \end{aligned} \tag{3.3}$$

Using the optimal coefficient c^*, the resulting variance is:

$$\begin{aligned}\text{Var}(X^*) &= \text{Var}(X) + \left(\frac{\text{Cov}(X, Y)}{\text{Var}(Y)}\right)^2 \text{Var}(Y) - 2\left(\frac{\text{Cov}(X, Y)}{\text{Var}(Y)}\right)\text{Cov}(X, Y) \\ \text{Var}(X^*) &= \text{Var}(X) - \frac{\text{Cov}(X, Y)^2}{\text{Var}(Y)} \\ \text{Var}(X^*) &= \text{Var}(X) - \frac{\text{Cov}(X, Y)^2}{\text{Var}(X)\text{Var}(Y)}\text{Var}(X) \\ \text{Var}(X^*) &= \big(1 - \text{Corr}(X, Y)^2\big)\text{Var}(X) \end{aligned} \tag{3.4}$$

where $\text{Corr}(X, Y)$ is the correlation coefficient of X and Y. From this, it can be seen that a larger magnitude of correlation between the two variables results in a greater reduction in variance.

3.2 Control Variates for TD Learning

To derive control variates for TD learning, we first look at the action-value setting. n-step Sarsa with per-decision importance sampling estimates the return based on the recursive definition in Equations 2.28 and 2.29:

$$\begin{aligned}\hat{G}_{t:t} &= Q(S_t, A_t) \\ \hat{G}_{t:t+n} &= R_{t+1} + \gamma\rho_{t+1}\hat{G}_{t+1:t+n}\end{aligned}$$

We focus on the importance sampling-corrected portion of the n-step return, $\rho_{t+1}\hat{G}_{t+1:t+n}$. Suppose that at each step, through the sampled action, the n-step Sarsa algorithm jointly samples the importance sampling-corrected action-value $\rho_{t+1}Q(S_{t+1}, A_{t+1})$. Since the target policy is typically known, the action-value of a particular state has a known expected value at a given time step, $\bar{Q}_{t+1}$. Using the form of the control variate in Equation 3.1, we get the

following estimate of the importance-sampling corrected term of the n-step return:

$$(\rho_{t+1}\hat{G}_{t+1:t+n})^* = \rho_{t+1}\hat{G}_{t+1:t+n} + c\big(\rho_{t+1}Q(S_{t+1}, A_{t+1}) - \bar{Q}_{t+1}\big) \tag{3.5}$$

Under the assumption that the approximated value function is accurate, the action-values represent the expected return. Due to this, the sampled reward sequence and the action-value are, in expectation, the same quantity. The covariance term in Equation 3.3 would be the variance of the sampled action-value, and from this, a reasonable choice for the coefficient would be -1. This gives:

$$(\rho_{t+1}\hat{G}_{t+1:t+n})^* = \rho_{t+1}\hat{G}_{t+1:t+n} + \bar{Q}_{t+1} - \rho_{t+1}Q(S_{t+1}, A_{t+1}) \tag{3.6}$$

Substituting this estimate into the recursive definition of n-step Sarsa's estimate of the return gives the following n-step return (Maintaining the same bootstrapping condition):

$$\hat{G}^{ACV}_{t:t+n} = R_{t+1} + \gamma(\rho_{t+1}\hat{G}_{t+1:t+n} + \bar{Q}_{t+1} - \rho_{t+1}Q(S_{t+1}, A_{t+1})) \tag{3.7}$$

The expectation of the additional terms under the behavior policy for a given state is 0, so no additional bias is introduced:

$$\begin{aligned}
&\mathbb{E}_\mu[\bar{Q}_{t+1} - \rho_{t+1}Q(S_{t+1}, A_{t+1})] \\
&\qquad = \sum_a \mu(S_{t+1}, a)\left(\bar{Q}_{t+1} - \frac{\pi(S_{t+1}, a)}{\mu(S_{t+1}, a)}Q(S_{t+1}, a)\right) \\
&\qquad = \bar{Q}_{t+1} - \sum_a \pi(S_{t+1}, a)Q(S_{t+1}, a) \\
&\qquad = \bar{Q}_{t+1} - \bar{Q}_{t+1} \\
&\qquad = 0
\end{aligned}$$

To provide intuition on how it might reduce the variance in the estimate, we can consider some extreme cases of the importance sampling ratio. If $\rho_{t+1} = 0$, when the behaviour policy takes an action that the target policy would have never taken, it will bootstrap off of the expectation of its current estimates instead of making the remainder of the reward sequence equal to

0. If ρ_{t+1} is much greater than 1, an equivalent amount of its current action-value estimate is subtracted to compensate. This can be seen as having the benefit of smaller products of importance sampling ratios from per-decision importance sampling, combined with the benefit of applying the importance sampling ratios to a difference between the sampled return and an estimate like in Equation 2.10.

In the one-step case, the introduction of this control variate results in one-step Expected Sarsa's target:

$$\hat{G}_{t:t+1} = R_{t+1} + \gamma(\rho_{t+1}Q(S_{t+1}, A_{t+1}) + \bar{Q}_{t+1} - \rho_{t+1}Q(S_{t+1}, A_{t+1}))$$
$$\hat{G}_t = R_{t+1} + \gamma\bar{Q}_{t+1}$$

When applying the control variate at the bootstrapping step, it implicitly results in bootstrapping off of the expectation under the target policy as opposed to the importance sampling-corrected action-value. From this, it can be viewed as an alternate generalization of Expected Sarsa to the multi-step setting. n-step Expected Sarsa would be a special case which only applies the control variate at the bootstrapping step, when it could be applied to each sampled reward in the sequence.

The control variate can be interpreted as performing an *expectation correction* at each step based on current estimates. TD methods aim to learn the expectation across all possible trajectories under a policy, but tries to do so with the immediate information from the trajectory currently being taken. An action-value is like a closer guess of what the current sampled return will be than the computed expectation across all actions, because the agent knows which action resulted in the immediate reward at each step. The difference between the expected action-value across all actions and the action-value of the action selected can be seen as a per-step estimate of the discrepancy between the expectation across all reward sequences, and the sampled reward sequence from the current step onwards.

These additional terms in the action-value setting can also be viewed as a model-free instance of a doubly robust estimator which has been studied in in offline methods for sequential decision making (Jiang et al., 2016; Thomas

et al., 2016). Harutyunyan et al. (2016) has explored similar expectation correction terms, but in the context of whether the terms could perform off-policy corrections without the use of importance sampling.

Looking at the state-value case, we start with the n-step return given by Equations 2.26 and 2.27:

$$\hat{G}_{t:t} = V(S_t)$$
$$\hat{G}_{t:t+n} = \rho_t(R_{t+1} + \gamma \hat{G}_{t+1:t+n})$$

Unlike the action-value setting, the algorithm doesn't have access to action-values which incorporate information about the immediate reward for the action selected. However, it can still take advantage of knowing what actions were taken through the importance sampling ratios. It may instead jointly sample the importance sampling-corrected state-value, $\rho_t V(S_t)$. This quantity has a known expected value of $V(S_t)$, since state-values already represent the expectation across all actions. This results in the following n-step return:

$$\hat{G}^{SCV}_{t:t+n} = \rho_t(R_{t+1} + \gamma \hat{G}_{t+1:t+n}) + c\big(\rho_t V(S_t) - V(S_t)\big) \tag{3.8}$$

Setting the control variate parameter c to -1 under the same assumptions made in the action-value setting, we get:

$$\hat{G}^{SCV}_{t:t+n} = \rho_t(R_{t+1} + \gamma \hat{G}_{t+1:t+n}) + V(S_t) - \rho_t V(S_t)$$
$$\hat{G}^{SCV}_{t:t+n} = \rho_t(R_{t+1} + \gamma \hat{G}_{t+1:t+n}) + (1 - \rho_t)V(S_t) \tag{3.9}$$

This has a similar intuition as the action-value setting, where knowledge of the action-selected allows it to have a closer guess of the current importance sampling-corrected reward sequence. For example, if the specific action selected results in a very large importance sampling ratio, the importance sampling-corrected state value would be a closer estimate to what the current sampled return will be. Taking the difference between the state value and the importance sampling-corrected state-value can be seen as estimating the discrepancy between the current sampled sequence and its guess of what the expected return is supposed to be. Similarly, the additional terms do not

introduce additional bias:

$$\begin{aligned}
&\mathbb{E}_\mu[V(S_{t+1}) - \rho_{t+1}V(S_{t+1})] \\
&\quad = \sum_a \mu(S_{t+1}, a)\left(V(S_{t+1}) - \frac{\pi(S_{t+1}, a)}{\mu(S_{t+1}, a)}V(S_{t+1})\right) \\
&\quad = V(S_{t+1}) - V(S_{t+1})\sum_a \pi(S_{t+1}, a) \\
&\quad = V(S_{t+1}) - V(S_{t+1}) \\
&\quad = 0
\end{aligned}$$

Of note, the state-value control variate disappears in the on-policy setting, while the action-value control variate does not. This makes the above control variates generally applicable to off-policy learning, but we did not explore further control variates which are generally applicable to the on-policy setting. Because state-values can be computed from action-values through Equation 2.4, the state-value control variate may also be applied to action-value methods.

3.3 Relationship with Existing Algorithms

Adding the action-value control variate into the recursive definition for the λ-return of the Sarsa(λ) algorithm in Equation 2.33, we get:

$$\begin{aligned}
\hat{G}_t^\lambda &= R_{t+1} + \gamma\big((1-\lambda)(\rho_{t+1}Q(S_{t+1}, A_{t+1}) + \bar{Q}_{t+1} - \rho_{t+1}Q(S_{t+1}, A_{t+1})) \\
&\quad + \lambda(\rho_{t+1}G_{t+1}^\lambda + \bar{Q}_{t+1} - \rho_{t+1}Q(S_{t+1}, A_{t+1}))\big) \\
\hat{G}_t^\lambda &= R_{t+1} + \gamma\big((1-\lambda)\bar{Q}_{t+1} + \lambda(\rho_{t+1}G_{t+1}^\lambda + \bar{Q}_{t+1} \\
&\quad - \rho_{t+1}Q(S_{t+1}, A_{t+1}))\big)
\end{aligned} \tag{3.10}$$

Solving the above recursion, we can rearrange it into a sum of one-step Expected Sarsa's TD errors:

$$\hat{G}_t^\lambda = Q(S_t, A_t) + \sum_{k=t}^{\infty}(R_{k+1} + \gamma\bar{Q}_{k+1} - Q(S_k, A_k))\prod_{i=t+1}^{k}\gamma\lambda\rho_i \tag{3.11}$$

This is equivalent to using the eligibility trace decay rate of Sarsa(λ), but backing up the TD errors of one-step Expected Sarsa. That is, in the space

of action-value TD(λ) algorithms, having the one-step estimates of the return bootstrap off of the expectation under the target policy implicitly induces this per-decision control variate in the corresponding n-step returns.

An aforementioned algorithm that also uses one-step Expected Sarsa's TD error in its λ-return is the Tree-backup(λ) algorithm. Tree-backup(λ)'s estimate of the return is characterized by Equation 2.35, re-stated below (Denoting $\pi_t = \pi(S_t, A_t)$):

$$\hat{G}_t^{\lambda} = Q(S_t, A_t) + \sum_{k=t}^{\infty} (R_{k+1} + \gamma \bar{Q}_{k+1} - Q(S_k, A_k)) \prod_{i=t+1}^{k} \gamma\lambda\pi_i$$

If we look at n-step Tree-backup's estimate of the return in Equation 2.31, and rewrite the summation over actions not taken as follows, we can show that it also includes similar expectation correction terms:

$$\hat{G}_{t:t+n} = R_{t+1} + \gamma\Big(\pi_{t+1}\hat{G}_{t+1:t+n} + \sum_{a \neq A_{t+1}} \pi(S_{t+1}, a)Q(S_{t+1}, a)\Big)$$
$$\hat{G}_{t:t+n} = R_{t+1} + \gamma\big(\pi_{t+1}\hat{G}_{t+1:t+n} + \bar{Q}_{t+1} - \pi_{t+1}Q(S_{t+1}, A_{t+1})\big)$$

The estimate takes some portion of the sampled reward sequence, and the difference between the expected action-value under the target policy and an equivalent portion of the sampled action-value. The algorithmic difference between n-step Tree-backup and n-step Sarsa with the per-decision control variate is the replacement of the π_t terms with the importance sampling ratios ρ_t. This *per-decision function*, which decides what portion of the sampled reward sequence to use, gets captured as an action-dependent trace decay rate.

Generalizing the above insight for the action-value setting, If we denote a TD algorithm's bootstrapping target as B_t, and some general per-decision function d_t, we get the following n-step return:

$$\hat{G}_{t:t} = Q(S_t, A_t)$$
$$\hat{G}_{t:t+n} = R_{t+1} + \gamma\big(d_{t+1}\hat{G}_{t+1:t+n} + B_{t+1} - d_{t+1}Q(S_{t+1}, A_{t+1})\big)$$

and corresponding λ-return:

$$\hat{G}_t^\lambda = R_{t+1} + \gamma\big((1-\lambda)B_{t+1} + \lambda(d_{t+1}\hat{G}_{t+1:t+n} + B_{t+1} - d_{t+1}Q(S_{t+1}, A_{t+1}))\big)$$
$$\hat{G}_t^\lambda = Q(S_t, A_t) + \sum_{k=t}^{\infty}(R_{k+1} + \gamma B_{k+1} - Q(S_k, A_k)) \prod_{i=t+1}^{k} \gamma\lambda d_i$$

For example, the Sarsa algorithm with per-decision importance sampling uses the bootstrapping target $B_t = \rho_t Q(S_t, A_t)$ and per-decision function $d_t = \rho_t$. Substituting these into the above equations will give the return estimates for the off-policy n-step Sarsa and Sarsa(λ) algorithms. This insight allows for for quickly mapping TD(λ) to their n-step TD counterparts, provided they follow a similar form. In the state-value setting, combining Equations 3.9 and 2.32 gives the following λ-return:

$$\hat{G}_t^\lambda = V(S_t) + \rho_t \sum_{k=t}^{\infty}(R_{k+1} + \gamma V(S_{k+1}) - V(S_k)) \prod_{i=t+1}^{k} \gamma\lambda\rho_i \tag{3.12}$$

which is an intuitive generalization of off-policy per-decision importance sampling for state-values, having an additional importance sampling correction term for the first reward in the sequence. It can be seen that scaling state-value TD errors by a per-decision function, as opposed to scaling the TD target, implicitly induces the state-value control variate in the n-step estimate of the return.

3.4 Experimental Results

In this section, we focus on the action-value setting and investigate the performance of n-step Sarsa with per-decision control variates on three problems. The first two are multi-step prediction tasks in a tabular environment, one being off-policy and one being on-policy. The remaining one is a control problem involving function approximation, evaluating the performance of n-step ACV Sarsa beyond the tabular setting, as well as how it handles a changing (greedifying) policy.

Since the introduction of the control variates can be viewed as another way of generalizing one-step Expected Sarsa to the multi-step, we compare the use

of control variates with the two existing multi-step generalizations: n-step Expected Sarsa and n-step Tree-backup. We test both the state-value control variate (denoted as *n-step SCV Sarsa*) and the action-value control variate (denoted as *n-step ACV Sarsa*). For n-step SCV Sarsa, the state-values are computed from the learned action-values using Equation 2.4, and it is made to bootstrap off of the expectation across action-values on the final step because the state-value control variate does not implicitly induce this.

3.4.1 5×5 Grid World

The *5×5 grid world* is a 2-dimensional grid world having terminal states in two opposite corners. The actions consist of 4-directional movement, and moving into a wall transitions the agent to the same state. The agent starts in the center, and a reward of -1 is received at each transition. Experiments were run in both the off-policy and on-policy settings with no discounting ($\gamma = 1$), and the root-mean-square (RMS) error between the learned value function and the true value function were compared:

$$RMSE = \sqrt{\frac{1}{|\mathcal{S}| \times |\mathcal{A}|} \sum_{s,a} (Q(s,a) - q_\pi(s,a))^2} \tag{3.13}$$

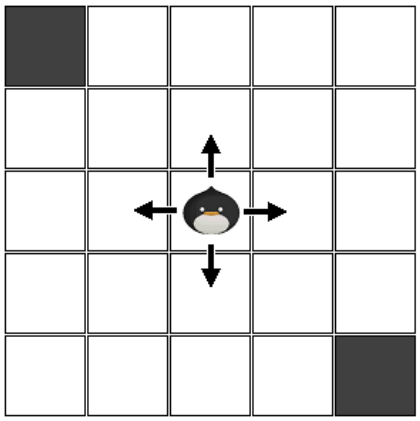

Figure 3.1: The 5×5 grid world environment. The top left and bottom right corners represent terminal states, and each transition results in a reward of -1. It was set up as an on and off-policy multi-step prediction task where the goal was to estimate the expected return under the target policy as accurately as possible.

Off-policy Prediction

For the off-policy experiments, the target policy would move north with probability $1 - \epsilon$, and select a random action equiprobably otherwise. ϵ was set to 0.5, and the behaviour policy was equiprobable random for all states. A parameter study was done for 1, 2, and 4 steps, and the RMS error was measured after 200 episodes. The results are averaged over 1000 runs, and can be seen in Figure 3.2.

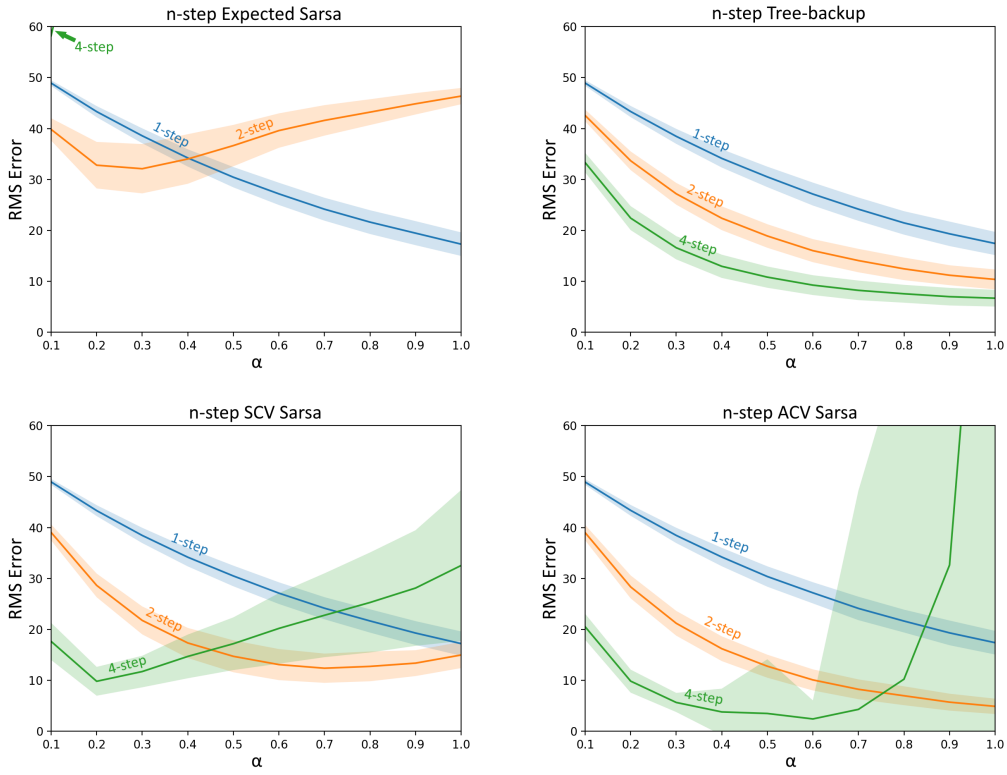

Figure 3.2: 5×5 grid world off-policy prediction results. This plot shows the performance of various parameter settings of each algorithm in terms of RMS error after 200 episodes in the learned value function. The shaded region corresponds to one standard deviation, and the results are averaged over 1000 runs.

It can be seen that 2-step Expected Sarsa only outperforms 1-step Expected Sarsa for a very limited range of parameters, but is worse otherwise. 4-step Expected Sarsa was unable to learn for most parameter settings. We can see that 2-step ACV Sarsa outperforms 1-step Expected Sarsa for all parameter settings, and the variance is reduced relative to 2-step Expected Sarsa. Fur-

thermore, 4-step ACV Sarsa ends up being able to learn, and can outperform 2-step ACV Sarsa for a reasonably wide range of parameters. n-step SCV Sarsa results in behavior in between that of n-step Expected Sarsa and n-step ACV Sarsa, but appears more stable for larger settings of n and α in the 4-step case. n-step Tree-backup has the lowest variance, and appears most stable at large parameter settings. However, Tree-backup learned relatively slowly that it did not achieve errors lower than the best settings of n-step ACV Sarsa within the 200 episodes.

Learning curves from the best tested parameter setting of each algorithm can be seen in Figure 3.3. It can be seen that n-step ACV Sarsa performs best on this problem. Despite having larger variance than the other algorithms at its best parameter setting, the control variate allowed ACV Sarsa to tolerate large enough combinations of n and α (out of those tested) to learn significantly faster. Of note, Tree-backup seems like it would perform better if we allowed n to take on larger parameter settings. However, we did not investigate this further as the primary focus was to investigate the effects of explicit introduction of control variates into n-step TD methods.

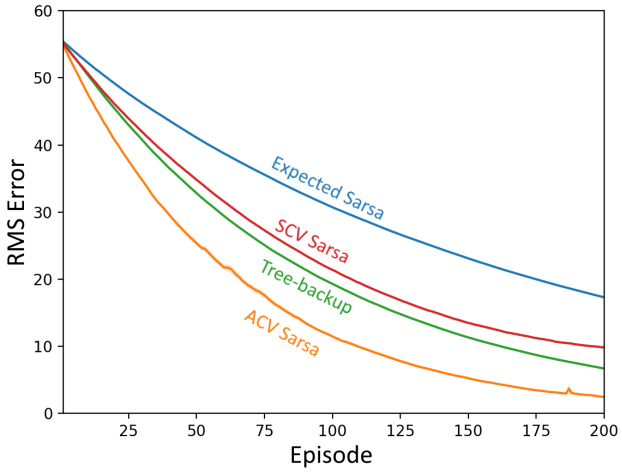

Figure 3.3: 5×5 grid world off-policy prediction learning curves. This plot shows the learning curve of the best parameter setting of each algorithm in terms of final RMS error after 200 episodes in the learned value function. The shaded region corresponds to one standard error, and the results are averaged over 1000 runs.

On-policy Prediction

In the on-policy case, the target policy and behaviour policy were both equiprobable random for all states. The parameters tested are identical to the off-policy experiment with the addition of 8-step instances of each algorithm. SCV Sarsa was not tested, as the state-value control variate is not present in the on-policy case. The RMS error was measured after 200 episodes, and are also averaged over 1000 runs. The results are summarized in Figure 3.4.

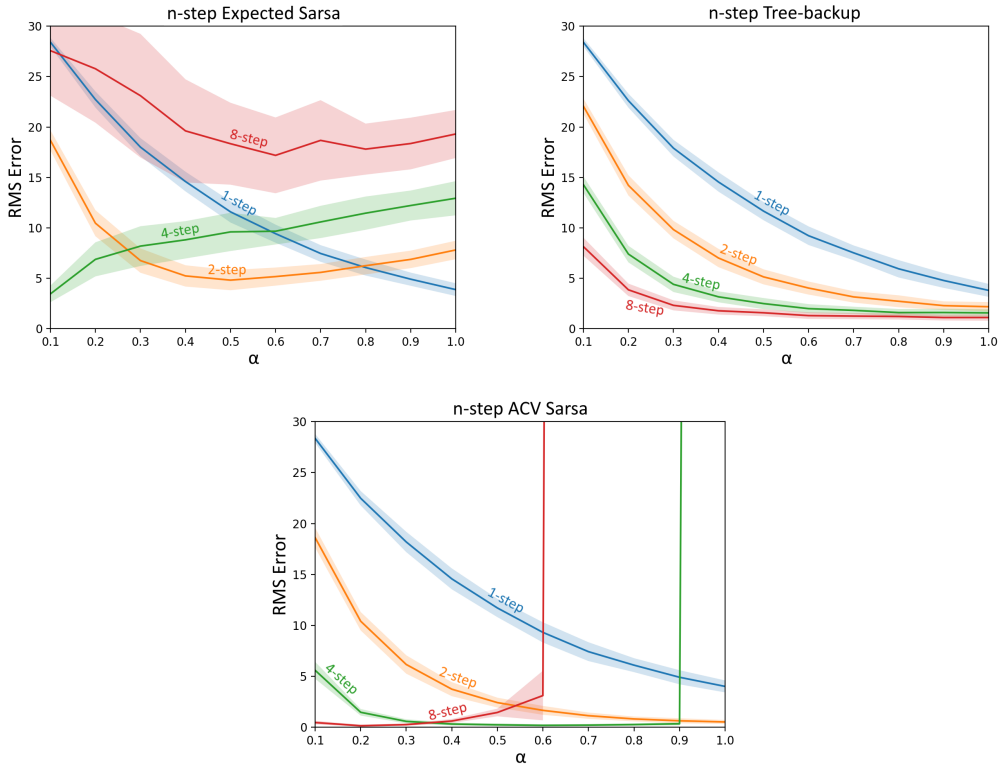

Figure 3.4: 5×5 grid world on-policy prediction results. This plot shows the performance of various parameter settings of each algorithm in terms of RMS error after 200 episodes in the learned value function. The shaded region corresponds to one standard deviation, and the results are averaged over 1000 runs.

2-step Expected Sarsa ends up performing better than 1-step Expected Sarsa for a wider range of parameters than in the off-policy case, but the best parameter settings for each perform similarly. Further increasing the number of steps results in relatively poor performance, and doesn't do better than the best parameter setting of 1-step Expected Sarsa. Looking at n-step ACV

Sarsa, we can see that performance is drastically improved for all tested settings of n. Of note, while introducing the per-decision control variate resulted in lower variance for a reasonable range of parameters, assumptions were made regarding the accuracy of the value function when setting the control variate parameter c in Equation 3.5. If the number of steps n and the step size α get too large, it can result in larger variance and divergence on parameter settings where n-step Expected Sarsa did not diverge. We did not investigate alternate methods of setting the control variate parameter in this work. Like in the off-policy case, Tree-backup was stable for all tested parameter settings, but did not outperform ACV Sarsa due to its relatively slow learning speed.

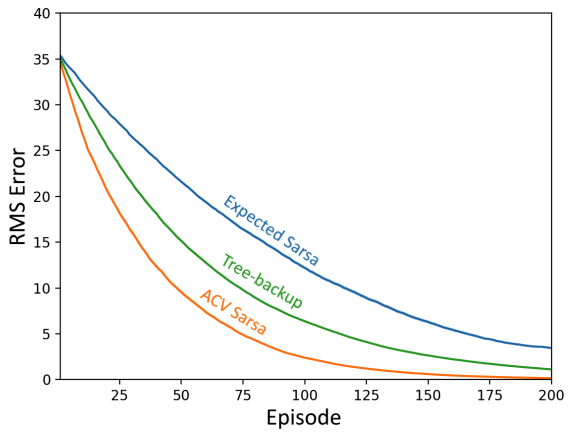

Figure 3.5: 5×5 grid world on-policy prediction learning curves. This plot shows the learning curve of the best parameter setting of each algorithm in terms of final RMS error after 200 episodes in the learned value function. The shaded region corresponds to one standard error, and the results are averaged over 1000 runs.

3.4.2 Mountain Car

To show that this use of control variates is compatible with function approximation, we ran experiments on *mountain car* as described by Sutton and Barto (2018). A reward of -1 is received at each step, and there is no discounting.

Because the environment's state space is continuous, we used *tile coding* (Sutton and Barto, 1998) to produce a feature representation for use with linear function approximation. The tile coder used 16 tilings, an asymmetric

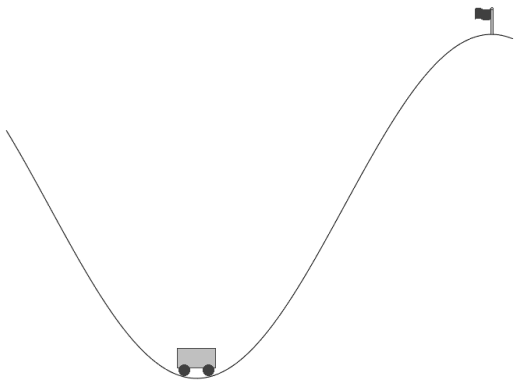

Figure 3.6: The mountain car environment (Sutton and Barto, 2018). The agent starts at a random location in the valley, receives a reward of -1 at each step, and its goal is to drive past the flag in as few steps as possible.

offset by consecutive odd numbers, and each tile covered 1/8-th of the feature space in each direction.

We compared n-step Expected Sarsa, n-step Tree-backup, and n-step ACV Sarsa with 1, 2, 4, and 8 steps across different step sizes α. Each algorithm learned on-policy with an ϵ-greedy policy which selects an action greedily with respect to its value function with probability $1-\epsilon$, and selected a random action equiprobably otherwise. In this experiment, ϵ was set to 0.1. We measured the return per episode up to 100 episodes, and averaged the results over 100 runs. Among the parameter settings tested, we found that $n = 4$ and $\alpha = 0.5$ was best for n-step Expected Sarsa and n-step ACV Sarsa, while $n = 8$ and $\alpha = 0.9$ was best for n-step Tree-backup. Figure 3.7 shows the results for the best parameter setting for each algorithm.

The two algorithms showed a similar trend in the parameters as in the 5×5 Grid World environment, but were less pronounced. This is likely due to not requiring accurate value function estimates to perform the task well, and the control variate having less of an effect with greedier target policies, because $\bar{Q}_{t+1}$ gets relatively close to $Q(S_t, A_t)$. Despite this, as seen in the results for the best parameter settings, n-step ACV Sarsa still outperforms n-step Expected Sarsa on this task. n-step Tree-backup performed similarly to n-step ACV Sarsa. This is possibly due to the ϵ-greedy policy being relatively less stochastic than the target policies used in the other domains, resulting in

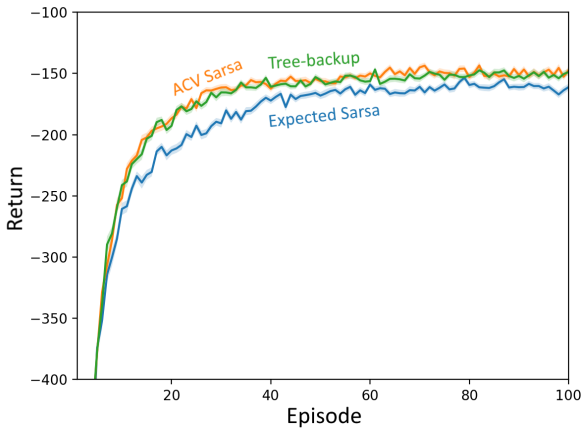

Figure 3.7: Mountain car on-policy control results. The plot shows the return per episode of the best parameter setting of each algorithm in terms of the mean return over all episodes. The shaded region corresponds to one standard error, and the results are averaged over 100 runs.

lower bias.

Chapter 4

A Unifying Algorithm

In this chapter, we present the n-step Q(σ) algorithm, an algorithm which unifies the space of n-step TD methods presented in Chapter 2 of this thesis. From being able to recover existing algorithms through various settings of the σ parameter, we can gain insight on properties of existing methods, as well as the potential benefit of combining them. We first provide some analysis and empirical results for this algorithm, then extend it with the per-decision control variates. Next, we derive the TD(λ) version of the algorithm with eligibility traces, the Q(σ, λ) algorithm, and show how this algorithm gives an alternate view of existing multi-step TD methods.

4.1 The Landscape of TD Control Methods

If we recall the space of one-step TD methods for action-values, there are two main algorithms. The first is one-step Sarsa, characterized by the TD target in Equation 2.19, and the second is one-step Expected Sarsa, characterized by the TD target in Equation 2.23. The difference between these two algorithms is the bootstrapping target. The Sarsa algorithm bootstraps off of the importance sampling-corrected action-value of the sampled action, and Expected Sarsa bootstraps off of the expected action-value under the target policy.

One-step Sarsa generalizes to the n-step Sarsa algorithm, which only uses information from the sampled action at each step. However, one-step Expected Sarsa has two notable extensions to the n-step setting. The first extension is the n-step Expected Sarsa algorithm, which uses information from the sampled

actions up until the n-th step, at which it bootstraps off of an expectation across all actions. The second multi-step extension is the n-step Tree-backup algorithm, which computes an expectation across all actions at each step. The exact definitions for the above algorithms can be found in Chapter 2.

In the one-step setting, computing the expectation is generally better as it avoids unnecessary variance due to policy stochasticity (van Seijen et al., 2009) without incurring additional bias. This isn't the case in the multi-step setting, where the reduced variance in computing an expectation may result in increased bias in the return estimate. This is evident when comparing the weight an n-step TD algorithm gives to the sampled reward sequence for a given number of steps n. For example, 4-step Sarsa will include 4 sampled rewards in its estimate, while 4-step Tree-backup effectively includes fewer than 4 rewards. We can show this by expanding n-step Tree-backup's estimate of the return in Equation 2.31 and only consider the terms involving the sampled rewards:

$$\hat{G}_{t:t+n} = R_{t+1} + \gamma\left(\pi_{t+1}\hat{G}_{t+1:t+n} + \sum_{a \neq A_{t+1}} \pi(S_{t+1}, a)Q(S_{t+1}, a)\right)$$
$$\hat{G}_{t:t+n} = R_{t+1} + \gamma\pi_{t+1}R_{t+2} + \gamma^2\pi_{t+1}\pi_{t+2}R_{t+3} + \gamma^3\pi_{t+1}\pi_{t+2}\pi_{t+3}R_{t+4} + ...$$

contrasting with n-step Sarsa, recalling that $\mathbb{E}_\mu[\rho_t|S_t = s] = 1$:

$$\hat{G}_{t:t+n} = R_{t+1} + \gamma\rho_{t+1}\hat{G}_{t+1:t+n}$$
$$\hat{G}_{t:t+n} = R_{t+1} + \gamma\rho_{t+1}R_{t+2} + \gamma^2\rho_{t+1}\rho_{t+2}R_{t+3} + \gamma^3\rho_{t+1}\rho_{t+2}\rho_{t+3}R_{t+4} + ...$$

This effective reduction in the number of steps arises from computing expectations at earlier steps, as doing so weights the remainder of the reward sequence by the probability of the action selected. This reduced emphasis on the sampled reward sequence is compensated for by bootstrapping off of the current estimates for the actions not taken, and this additional reliance on current estimates to be accurate results in larger bias. This shows a bias-variance tradeoff between using sampled information and computing expectations in the multi-step setting, and motivates the unification of these methods where an algorithm can take advantage of the benefits of either extreme.

4.2 The n-step Q(σ) algorithm

The n-step Q(σ) algorithm was initially proposed by Precup et al. (2000), and unifies the aforementioned n-step TD methods for action-values. The algorithm is characterized by deciding at each step whether to use information from the sampled action, compute an expectation across all actions, or a mixture of the two. This decision is controlled by a hyperparameter $\sigma \in [0, 1]$, and has the following recursive definition:

$$\hat{G}_{t:t} = Q(S_t, A_t) \tag{4.1}$$

$$\begin{aligned}\hat{G}_{t:t+n} = R_{t+1} + \gamma\Bigg(&\sigma_{t+1}\rho_{t+1}\hat{G}_{t+1:t+n} \\ &+ (1 - \sigma_{t+1})\Bigg(\pi_{t+1}\hat{G}_{t+1:t+n} + \sum_{a \neq A_{t+1}} \pi(S_{t+1}, a)Q(S_{t+1}, a)\Bigg)\Bigg)\end{aligned} \tag{4.2}$$

Considering fixed values of σ, setting $\sigma = 0$ results in the n-step Tree-backup algorithm, which commits to computing an expectation at each step. On the $\sigma = 1$ extreme we get the n-step Sarsa algorithm, which uses information from the sampled action at each step. Setting σ to fixed intermediate values between these two extremes can be viewed as computing a weighted average of n-step Sarsa and n-step Tree-backup's TD targets. The parameter σ_t may also be set dynamically. If we set $\sigma_t = 1$ for steps $t + 1$ to $t + n - 1$, and then set $\sigma_{t+n} = 0$ for the bootstrapping step, we get the n-step Expected Sarsa algorithm.

We can develop a simple heuristic approach for dynamically varying the parameter based on intuition from the bias-variance tradeoff, and how the bias relies on current estimates being accurate. The algorithm can start with $\sigma = 1$, which has large variance but less reliance on estimates being correct. Assuming the estimates are improving over time, we can gradually decrease σ over time to approach $\sigma = 0$, the extreme that has lower variance, but more reliance on accurate estimates. One way to implement this in an episodic setting would be to multiply σ by some factor $\beta \in [0, 1)$ after each episode.

4.3 n-step Q(σ) Results

In this section we provide emprical results on the n-step Q(σ) algorithm, showcasing the potential benefit of trading off between sampling and computing expectations in multi-step TD learning. We additionally compare the aformentioned heuristic approach for dynamically varying σ, and show that it can outperform fixed values. In each experiment, *dynamic* σ refers to applying a decay factor $\beta = 0.95$ after each episode.

4.3.1 19-State Random Walk

The *19-state random walk*, shown in Figure 4.1, is a 1-dimensional environment where an agent randomly transitions to one of two neighboring states. There is a terminal state on each end of the environment, transitioning to one of them gives a reward of -1, and transitioning to the other gives a reward of 1. The task is formulated such that each state has two actions, and each action deterministically transitions to one of the two neighboring states. The agent then learns on-policy under an equiprobable random behavior policy.

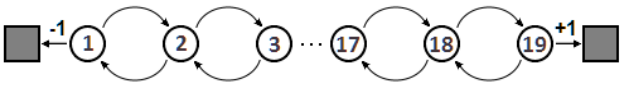

Figure 4.1: The 19-state random walk environment. It was set up as an on-policy prediction task with an agent behaving under an equiprobable random behavior policy.

This environment was treated as a prediction task, and the RMS error between the learned value function and the true value function was computed after each episode. For fixed values of σ, as well as a dynamic σ, we performed a sweep across various backup-lengths n and step sizes α. Each $Q(\sigma)$ instance and parameter setting was run for 50 episodes and the results are averaged over 100 runs.

Figure 4.2 shows the results with $n = 3$ and $\alpha = 0.4$, which was found to be representative of the best parameter setting for each instance of $Q(\sigma)$ on this task. Sarsa (full sampling) had better initial performance but poor asymptotic

performance, Tree-backup (no sampling) had poor initial performance but better asymptotic performance, and intermediate degrees of sampling traded off between the initial and asymptotic performances. This supports the intuition of the afformentioned bias-variance tradeoff, and it can be seen that the dynamic σ outperforms all of the fixed values of σ tested.

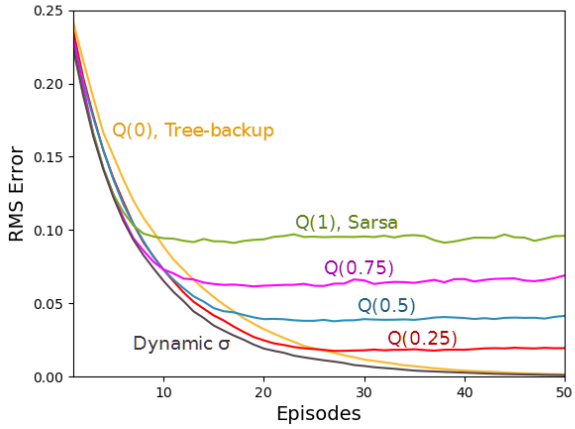

Figure 4.2: 19-state random walk prediction results. This plot shows the performance of $Q(\sigma)$ in terms of RMS error in the value function. The results are an average of 100 runs, and the standard errors are all less than 0.006. $Q(1)$ had the best initial performance, $Q(0)$ had the best asymptotic performance, and the dynamic σ outperformed all fixed values of σ.

4.3.2 Stochastic Windy Grid World

The *windy gridworld* is a tabular control task described by Sutton and Barto (2018), and it takes place in a 2-dimensional grid world. The actions consist of 4-directional movement, and moving into a wall transitions the agent to the same state. There is a start state and a goal state, and the agent is affected by column-dependent *wind* which shifts the resultant next state upward by a number of cells. The agent receives a reward of -1 at each time step, encouraging the agent to reach the goal state in as few steps as possible. The layout of the windy gridworld problem can be seen in Figure 4.3.

The original windy gridworld domain was deterministic, where a given state-action pair had a single possible outcome. We implemented a *stochastic*

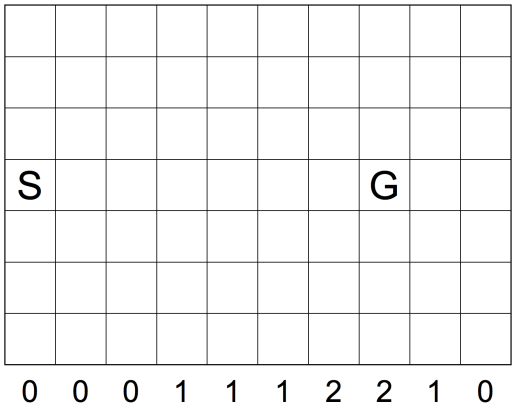

Figure 4.3: The windy gridworld environment. The start and goal states are denoted by S and G respectively, and the numbers underneath each column denote the upward shift in the resultant next state when transitioning into that column.

windy gridworld, where after each action, there is a 10% chance for the next state to be selected at random from one of the 8 states surrounding the agent.

We ran an experiment on the stochastic windy gridworld to compare various instances of the Q(σ) algorithm in a control setting. Each Q(σ) instance learned on-policy under an ϵ-greedy policy with $\epsilon = 0.1$, and no discounting was used ($\gamma = 1$). We performed 1000 runs of 100 episodes, and the average return over the 100 episodes was computed for each run. For fixed values of σ, as well as a dynamic σ, we tested various backup-lengths n and step sizes α. The results are summarized in Figure 4.4.

For all settings of σ that we tested, $n = 3$ resulted in the largest average return per episode. From the results, it can be seen that intermediate values of σ can outperform either extreme, and the dynamic σ performed best overall.

4.3.3 Mountain Cliff

We implemented a variant of mountain car, denoted as *mountain cliff*. In this variant, if the agent were to go too far to the left, it would fall off a cliff, and be returned to a random initial location in the valley with a reward of -100. All other aspects of the task are identical to that of mountain car as described by Sutton and Barto (2018). Both mountain car and mountain cliff environments

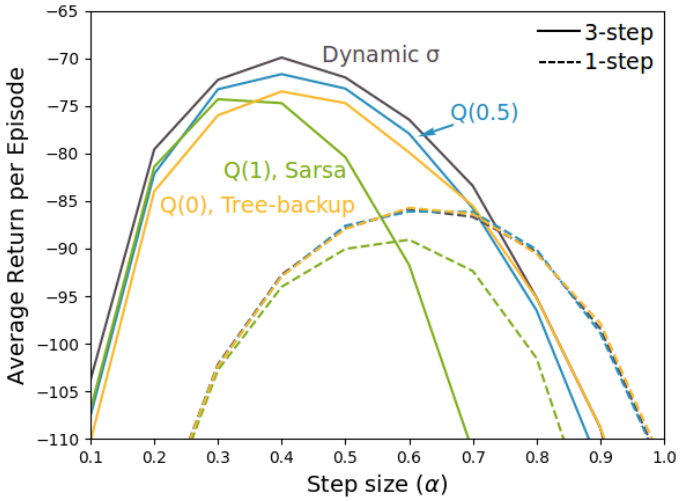

Figure 4.4: Stochastic windy gridworld results. The plot shows the performance of $Q(\sigma)$ in terms of the average return over 100 episodes as a function of the step size, α, for various values of σ. The results are for selected α values, then are connected by straight lines, and are an average of 1000 runs. The standard errors are all less than 0.3 which is about a line width. 3-step algorithms performed better than their 1-step equivalents, and $Q(\sigma)$ with a dynamic σ performed the best overall.

were tested, and they showed similar trends in the results. However, the results obtained on mountain cliff are more pronounced, and thus were more suitable for demonstrating the Q(σ) algorithm.

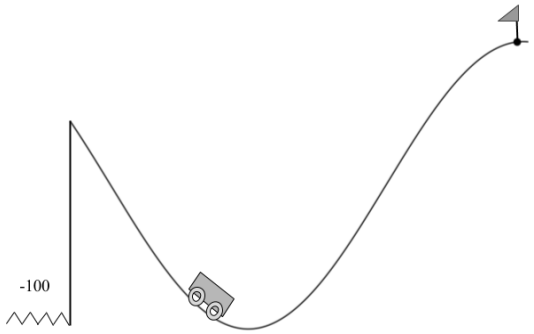

Figure 4.5: The mountain cliff environment. The agent starts at a random location in the valley, receives a reward of -1 at each step, and falling off the cliff returns it to a random initial location in the valley with a reward of -100. The goal is to drive past the flag in as few steps as possible.

We approximated the continuous state space with tile coding function ap-

proximation. The tile coder used 32 tilings, an asymmetric offset by consecutive odd numbers, and each tile covered 1/8-th of the feature space in each direction. For each Q(σ) instance, we performed 500 runs of 500 episodes each. Each Q(σ) instance learned on-policy under an ϵ-greedy policy with $\epsilon = 0.1$, and no discounting was used ($\gamma = 1$). We performed 1500 runs of 500 episodes, and the average return over the 500 episodes was computed for each run. For each setting of σ, including a dynamic σ, we optimized the back-up length n and step size α for the largest average return over the 500 episodes. The results of the best parameter settings found for each setting of σ are summarized in Figure 4.6.

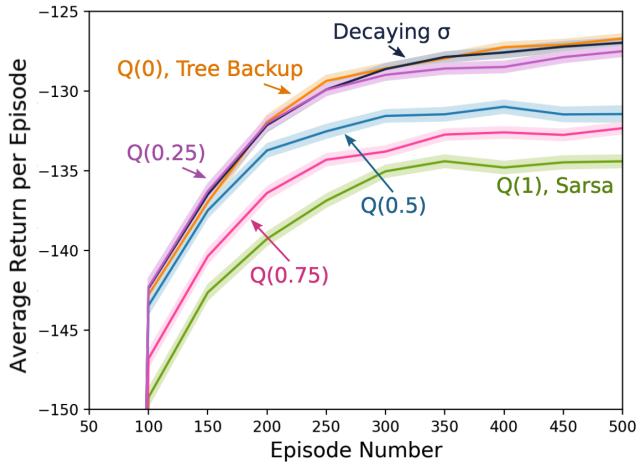

Figure 4.6: Mountain cliff results. This plot shows the performance of four Q(σ) instances with their best parameter settings in terms of average return over 500 episodes. The results are smoothed with a right-centered moving average with a window size of 50. The shaded region corresponds to a 95% confidence interval with a t-distribution.

From the results, dynamic σ outperformed all of the fixed settings of σ with an average return per episode of -152.56. Of note, among the fixed values of σ, Q(0.5) had the best early performance than the other instances with an average return over the first 50 episodes of -352.88. These algorithms may be capable of better early or final performance, but the parameter settings were optimized for the average return across all 500 episodes.

4.4 n-step Q(σ) with Control Variates

Incorporating the control variates from Chapter 3 into the sampling extreme ($\sigma = 1$) of the n-step Q(σ) algorithm, Equation 4.2 becomes:

$$\begin{aligned}
\hat{G}^{CV}_{t:t+n} &= R_{t+1} + \gamma\Bigg(\sigma_{t+1}\big(\rho_{t+1}\hat{G}_{t+1:t+n} + \bar{Q}_{t+1} - \rho_{t+1}Q(S_{t+1}, A_{t+1})\big) \\
&\quad + (1-\sigma_{t+1})\Bigg(\pi_{t+1}\hat{G}_{t+1:t+n} + \sum_{a \neq A_{t+1}} \pi(S_{t+1}, a)Q(S_{t+1}, a)\Bigg)\Bigg) \\
\hat{G}^{CV}_{t:t+n} &= R_{t+1} + \gamma\big(\sigma_{t+1}\big(\rho_{t+1}\hat{G}_{t+1:t+n} + \bar{Q}_{t+1} - \rho_{t+1}Q(S_{t+1}, A_{t+1})\big) \\
&\quad + (1-\sigma_{t+1})\big(\pi_{t+1}\hat{G}_{t+1:t+n} + \bar{Q}_{t+1} - \pi_{t+1}Q(S_{t+1}, A_{t+1})\big)\big) \\
\hat{G}^{CV}_{t:t+n} &= R_{t+1} + \gamma\big((\sigma_{t+1}\rho_{t+1} + (1-\sigma_{t+1})\pi_{t+1})\hat{G}_{t+1:t+n} + \bar{Q}_{t+1} \\
&\quad - (\sigma_{t+1}\rho_{t+1} + (1-\sigma_{t+1})\pi_{t+1})Q(S_{t+1}, A_{t+1})\big) \qquad (4.3)
\end{aligned}$$

For fixed values of σ, this resulting algorithm (denoted as n-step CV Q(σ)) can be seen as performing a weighted average between n-step ACV Sarsa and n-step Tree-backup. Of note, this algorithm can no longer recover n-step Sarsa. Doing so would have required a parameter to interpolate between a sample and expectation, and an additional hyperparameter to interpolate in the direction of control variates. For simplicity, the above algorithm only considers the latter, interpolating between a control-variate adjusted sampled action, and the expectation across all actions. From the results in Chapter 3, n-step ACV Sarsa had improved final performance on prediction problems relative to n-step Expected Sarsa, so n-step CV Q(σ) has less emphasis on a tradeoff between initial and final performance. However, n-step ACV Sarsa suffered from instability for larger parameter settings, while n-step Tree-backup was stable for all tested parameter settings. From this, n-step CV Q(σ) can be viewed as having more emphasis on balancing stability or parameter insensitivity.

We performed an experiment on the 19-state random walk as described in Section 4.3.1. For fixed values of σ, we performed a sweep across various backup-lengths n and step sizes α. Results comparing the parameter study between n-step Q(σ) and n-step CV Q(σ) for $\sigma = 0.75$ and $\sigma = 1$ can be seen in Figure 4.7

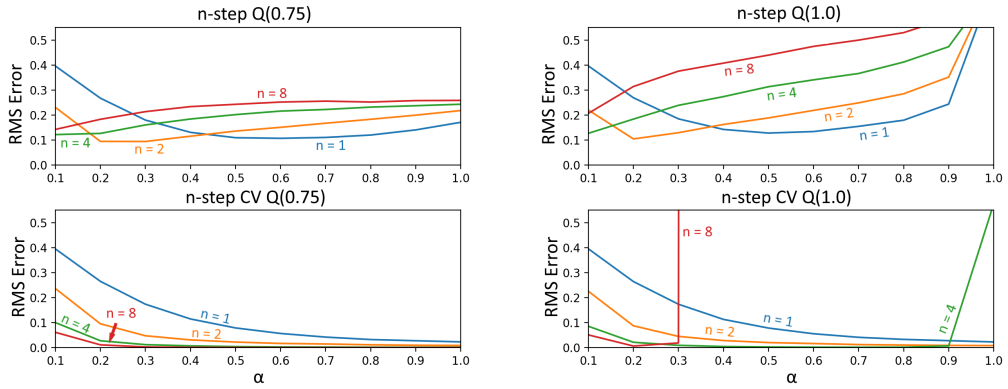

Figure 4.7: n-step CV Q(σ) comparison. This plot shows the performance of various parameter settings of n-step Q(σ) and n-step CV Q(σ) in terms of RMS error after 50 episodes in the learned value function. The results are averaged over 100 runs.

We chose to only show larger values of σ, as both algorithms are equivalent to Tree-backup at $\sigma = 0$. From the results, we can see that while n-step CV Q(σ) has improved final performance, it may suffer instability for large parameter settings. However, apparent from the $\sigma = 0.75$ plots, mixing a small portion of Tree-backup can address this instability without impacting the final performance. These results support the initial intuitions regarding mixtures of n-step ACV Sarsa and n-step Tree-backup.

4.5 The Q(σ, λ) Algorithm

Extending the n-step Q(σ) for use with eligibility traces results in the Q(σ, λ) algorithm. Using the version of the algorithm with control variates applied, n-step CV Q(σ), Equation 4.3 results in the following recursive definition of its λ-return:

$$\hat{G}_t^{\lambda} = R_{t+1} + \gamma\big(\lambda\bar{Q}_{t+1} + (1-\lambda)\big((\sigma_{t+1}\rho_{t+1} + (1-\sigma_{t+1})\pi_{t+1})\hat{G}_{t+1}^{\lambda} + \bar{Q}_{t+1} - (\sigma_{t+1}\rho_{t+1} + (1-\sigma_{t+1})\pi_{t+1})Q(S_{t+1}, A_{t+1})\big)\big) \tag{4.4}$$

Solving the recursion results in:

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma \bar{Q}_{t+1} - Q(S_t, A_t) \\ \hat{G}_t^\lambda &= Q(S_t, A_t) + \sum_{k=t}^{\infty} \delta_k \prod_{i=t+1}^{k} \gamma\lambda\big(\sigma_i \rho_i + (1-\sigma_i)\pi_i\big)\end{aligned} \tag{4.5}$$

Based on the insight from the control variates in Chapter 3, we get a sum of one-step Expected Sarsa's TD errors as opposed to the TD error of one-step Q(σ)'s error. It can be seen that the decision to sample or compute an expectation appears as a manipulable action-dependent trace decay rate. The idea of action-dependent trace decay rates isn't new, as methods like Sarsa(λ) had a trace decay rate dependent on the importance sampling ratio (If we allow λ_t to be larger than 1 and denote $\lambda_t = \lambda\rho_t$). Mahmood et al. (2017) has also investigated a general action-dependent bootstrapping framework. However, through its corresponding n-step return, the Q(σ, λ) algorithm gives an alternate view of action-dependent trace decay rates as managing a trade off between sampling and computing expectations. For example, the Retrace(λ) algorithm (Munos et al., 2016) is characterized by the following TD target:

$$\hat{G}_t^\lambda = Q(S_t, A_t) + \sum_{k=t}^{\infty} \big(R_{k+1} + \gamma \bar{Q}_{k+1} - Q(S_k, A_k)\big) \prod_{i=t+1}^{k} \gamma\lambda \min(1, \rho_i) \tag{4.6}$$

This can be seen as an instance of the Q(σ, λ) algorithm where σ is applied dynamically:

$$\sigma_t = \begin{cases} 1, & \text{if } \rho_t \leq 1 \\ \frac{1-\pi_t}{\rho_t - \pi_t}, & \text{if } \rho_t > 1 \end{cases}$$

This gives Retrace(λ) an interpretation of committing to using sampled information if the importance sampling ratio is small, then mix in a certain amount of the expected action value under the target policy when the importance sampling ratio is large.

Testing fixed values for σ in the Q(σ, λ) algorithm resulted in very similar plots to the n-step CV Q(σ) results in Figure 4.7 with λ in place of n. Due to this, we have omitted experimental results on this algorithm.

Chapter 5

Predicting Periodicity with Complex Value Functions

Following this thesis' theme of generalizing existing methods to provide deeper insight, this chapter introduces an alternative view of the discount rate used in the definition of the return. We focus on a specific relaxation of the range of values the discount rate can take on, where it may take on complex numbers. It can be shown that the resulting complex returns and learned values have an interpretation as identifying periodicity in the reward sequence. This extends the types of knowledge representable by value functions into the frequency domain, with the original setting of the discount rate representing the zero frequency case.

5.1 Complex Discounting

The discount rate has an interpretation of specifying the horizon of interest for the return, trading off between focusing on immediate rewards and considering the sum of longer sequences of rewards. It can also be interpreted as a *soft termination* of the return (Sutton et al., 2011), where an agent includes the next reward with probability γ, and terminates with probability $1-\gamma$, receiving a terminal reward of 0. From these interpretations, it is intuitive for the discount rate to fall in the range of $\gamma \in [0, 1)$ with the exception of episodic problems, where γ can be equal to 1.

With considerations for convergence, assuming the reward sequence is

bounded, restricting the discount rate to be in the range $\gamma \in [0, 1)$ makes the infinite sum (in the continuing case) of Equation 2.1 finite. However, this sum will be finite as long as the magnitude of the discount rate is restricted to be $|\gamma| \in [0, 1)$, allowing for the use of negative discount rates up to -1, as well as complex discount rates within the complex unit circle.

While the use of alternative discount rates may result in some corresponding value function, a question arises regarding whether these values are meaningful, or if there is any situation in which an agent would benefit from this knowledge. First, we consider the implications of exponenting a complex discount rate. We look at the exponential form of a complex number with unit magnitude, and acknowledge that it can be expressed as a sum of sinusoids by Euler's Formula:

$$e^{-i\omega} = \cos(\omega) - i\sin(\omega) \tag{5.1}$$

From this, it is evident that exponentiating a complex number to the power of n correponds to taking n steps around the complex unit circle with an angle of ω:

$$e^{-i\omega n} = \cos(n\omega) - i\sin(n\omega) \tag{5.2}$$

Using the above as a discount rate, assuming an episodic setting as it has a magnitude of 1, we would get the following return for some angle ω:

$$G_t^{\omega} = \sum_{k=0}^{T-t-1} e^{-i\omega k} R_{t+k+1} \tag{5.3}$$

Instead of weighting the reward sequence in a way that decays the importance of future rewards, complex discount rates weight the sequence with two sinusoids, one along the real axis and one along the imaginary axis. This can be interpreted as checking the cross correlation between a reward sequence and a sinusoid oscillating with a frequency of $\omega \frac{\text{rad}}{\text{step}}$, and effectively allows a TD learning agent to identify periodicity in the reward sequence at specified frequencies online and incrementally.

5.2 The Discrete Fourier Transform

The ability to identify periodicity in the reward sequence by weighting it with exponentiated complex numbers can be viewed as performing the Discrete Fourier Transform (DFT) from digital signal processing literature. The DFT is defined as follows:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi \frac{k}{N} n} \tag{5.4}$$

where N is the length of the sequence, and k is set to each whole number less than N. This can be viewed as testing whether a frequency of $2\pi \frac{k}{N}$ exists in the sequence, for equally spaced values of k. Acknowledging that k is less than N, the $\frac{k}{N}$ term is in the range $[0, 1)$, and can be rewritten where the frequency is specified directly:

$$X_\omega = \sum_{n=0}^{N-1} x_n e^{-i\omega n} \tag{5.5}$$

where $\omega \in [0, 2\pi)$. This is exactly equivalent to the DFT when the length of the sequence is known and particular frequencies ω are chosen, but has similar functionality and interpretation for other sets of frequencies.

The DFT corresponds to the discrete form of the coefficients of a Fourier series, and thus each complex number encodes the amplitude and phase of a particular sinusoidal component of a sequence. Specifically, the magnitude $|X_\omega|/N$ corresponds to the amplitude, and $\angle X_\omega = \arctan \frac{\mathrm{Im}(X_\omega)}{\mathrm{Re}(X_\omega)}$ gives the phase.

Applying this to the approximate values learned by a TD learning agent with a complex discount rate, it can be seen as computing the *expected DFT* of the reward sequence from a specified state onwards, and can extract the corresponding amplitude and phase information. However, the expected length of the sequence is typically not known by the agent, resulting in unnormalized amplitude information. Of note, the frequency is normalized by the agent's sampling frequency. That is, the frequency $\omega = 2\pi$ corresponds to the rate at which the agent is sampling rewards.

5.3 Revisiting Continuing Problems

The DFT is computed with complex numbers that have a magnitude of 1, and thus using similar complex discount rates would only work in the episodic setting. To see the effects of using a complex discount rate with a magnitude less than 1, we introduce an amplitude parameter $A \in [0, 1)$ to the exponential form of a complex number:

$$\gamma = Ae^{-i\omega} \tag{5.6}$$

which results in a complex number with a magnitude of A. Substituting this in the summation in Equation 2.1 gives:

$$G_t^\omega = \sum_{k=0}^{T-t-1} e^{-i\omega k} A^k R_{t+k+1} \tag{5.7}$$

which can be seen as computing the DFT of a discounted reward sequence with a discount rate of A. That is, a TD learning agent would still learn an expected DFT, but of the discounted return that is typically used in continuing settings.

5.4 Experimental Results

In this section we look at a simple motivating example to verify the affomentioned interpretations of complex value functions.

5.4.1 Checkered Grid World

The *checkered grid world* environment takes place on a 5×5 grid world with terminal states in the top left and bottom right corners. The actions consist of 4-directional movement, and moving into a wall transitions the agent to the same state. The agent starts in the center, and the board is colored with a checkerboard pattern with colors representing the reward distribution. Transitioning into a green cell results in a reward of 1, transitioning into a red cell results in a reward of -1, and transitioning to a terminal state ends the episode with a reward of 11. A diagram of the environment can be seen in Figure 5.1.

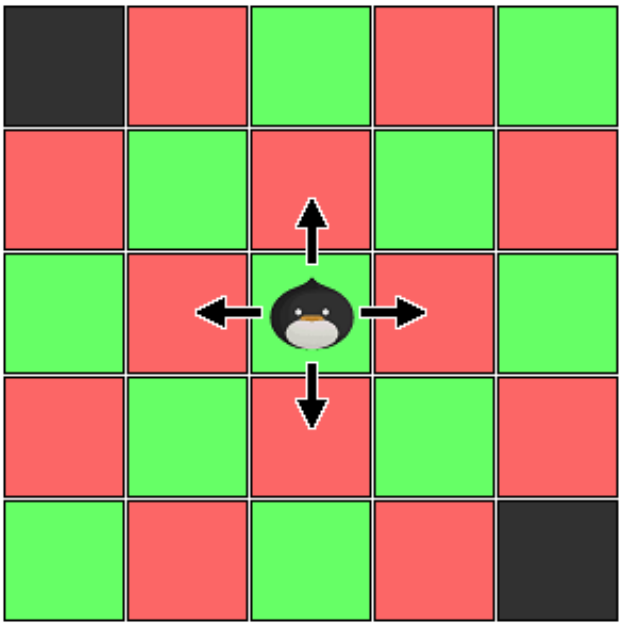

Figure 5.1: The checkered grid world environment. Transitioning into to a green square results in a reward of 1, and transitioning into a green square results in a reward of -1. The top left and bottom right corners represent terminal states where transitioning into them results in a reward of 11. It was set up as an on-policy prediction task with an agent behaving under an equiprobable random behavior policy.

This environment was treated as an on-policy prediction task with no discounting ($|\gamma| = 1$), and the agent learned many complex value functions in parallel. Specifically, it learned 100 value functions corresponding to equally spaced frequences in the range $\omega \in [0, 2\pi)$. The agent behaved under an equiprobably random behavior policy, and action-values were learned using the Tree-backup(λ) algorithm. State-values were computed from the learned action-values through Equation 2.4. We performed 100 runs of 200 episodes, and the value of the starting state, represented by the complex number's magnitude and phase information, was plotted for each frequency after the 200th episode. The resulting learned DFT of the starting state can be seen in Figure 5.2.

In the learned DFT, the magnitude of the value at $\omega = 0$ corresponds to the expected return with a discount rate of $\gamma = e^{-i0} = 1$. That is, it is what a "typical" TD learning agent with a non-oscillatory discount rate

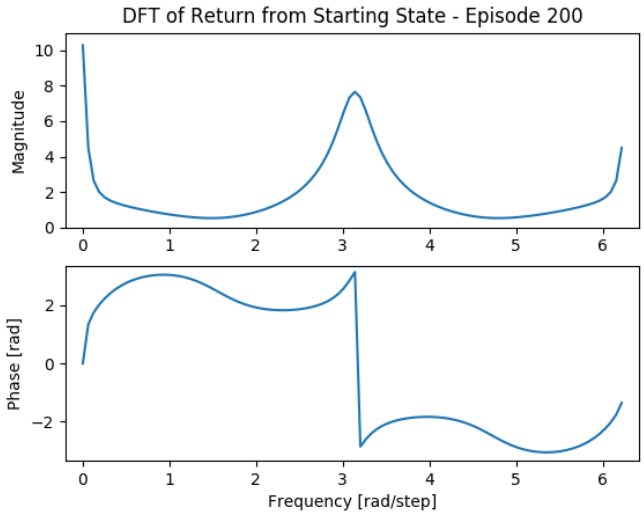

Figure 5.2: Checkered grid world results. The magnitude of the $\omega = 0$ point corresponds to the "actual" expected return under the behavior policy. The magnitudes of the other frequencies are interpreted as a measure of how confident the agent is that a particular frequency exists in the reward sequence. Results are averaged over 100 runs, and standard errors are less than the line width.

would have learned. The magnitudes of the values at other frequencies are interpreted as a measure of confidence in a particular frequency existing in the reward sequence, as exact amplitude information would require knowledge of the expected length of the sequence. It can be seen that there is relatively large magnitude at the frequency $\omega = \pi$, which corresponds to half of the agent's sampling frequency. If the agent is sampling at a rate of 1 sample per time step, this means that it is detecting an oscillation at a rate of half a cycle per time step. This corresponds to the rewards alternating between 1 and -1 in the environment, as this pattern takes two time steps to complete a cycle.

Chapter 6

Conclusions

This thesis was motivated by attempts to draw connections between existing multi-step TD methods, and develop strong intuitions about their properties. Having such a unified view could provide insight in how to effectively apply them in practice, as well as create new algorithms.

6.1 Summary

We looked at a control variate approach to address the large variance issue of off-policy learning in n-step TD methods. Their benefits were demonstrated empirically, and we used the resulting algorithms to provide insight about how n-step TD methods map to their TD(λ) counterparts. We then provided analysis and empirical results on an existing proposal to unify the space of n-step TD methods, the n-step Q(σ) algorithm. We further extended the n-step Q(σ) algorithm to use the afformentioned control variates, and then derived the TD(λ) version of the resulting algorithm: Q(σ, λ). Lastly, we showed how alternate settings of the discount rate parameter can allow for online and incremental learning of a signal's frequency information. This provides a new direction to explore in the space of TD methods, and an alternate view where the standard discount rate settings exist as the zero frequency case.

6.2 Future Directions

Attempting to bridge the gap between different algorithms often provides new insight regarding the algorithms, as well as new ideas to explore. Below we list some future directions which may follow from the work presented:

1. **Studying the Control Variate Parameter**: In our work, we made assumptions regarding the accuracy of the current value function estimates when setting the control variate parameter c. While this setting worked well for a reasonable range of back-up lengths and step sizes across several domains, it resulted in instability for some larger parameter combinations. Notably, the same large parameter combinations were sometimes stable for an equivalent algorithm without the control variates. Studying this instability may provide insight into a better approach for setting the parameter, such as one which adapts it dynamically.

2. **Improved Dynamic σ**: We presented a simple heuristic method of dynamically varying the σ parameter in the n-step Q(σ) algorithm. We later showed how existing algorithms like Retrace(λ) can also be interpreted as dynamically varying σ in the Q(σ, λ) algorithm. This direction has been investigated in the action-dependent bootstrapping framework by Mahmood et al. (2017), but perhaps contextualizing existing methods within the view of trading off between sampling and computing expectations may lead to new ways of dynamically varying the σ parameter.

3. **Application of Complex Value Functions**: By allowing the discount rate to take on complex numbers, we showed that a TD learning agent could incrementally estimate the expected DFT of the return. Work has been done on using the Fourier basis as a representation in reinforcement learning problems (Konidaris, 2008), that complex value functions may have a place as a learnable state representation. Furthermore, the digital signal processing community makes use of an inverse DFT to recover the origninal signal from the complex values. It may be possible to recover the exact reward sequence given the right set of frequencies to learn

about, allowing for more flexibility compared to an agent which only has access to the sum of the rewards.

References

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton.

De Asis, K., Hernandez-Garcia, J. F., Holland, G. Z., Sutton, R. S. (2018). Multi-step Reinforcement Learning: A Unifying Algorithm. In *Proceedings of The 32nd AAAI Conference on Artificial Intelligence*.

De Asis, K., Bennett, B., and Sutton, R. S. (2018). Predicting Periodicity with Temporal Difference Learning. *Under review*.

De Asis, K., Sutton, R. S. (2018). Per-decision Multi-step Temporal Difference Learning with Control Variates. In *Proceedings of The Conference on Uncertainty in Artificial Intelligence 2018*.

Hallak, A., Tamar, A., Munos, R., and Mannor, S. (2015). Generalized emphatic temporal difference learning: Bias-variance analysis. *arXiv:1509.05172*.

Hammersley, J. M, and Handscomb, D. C. (1964). *Monte Carlo Methods*. Methuen & Co. Ltd., London.

Harutyunyan, A., Bellemare, M. G., Stepleton, T., and Munos, R. (2016). Q(λ) with off-policy corrections. *arXiv:1509.05172*

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

Jaakola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation 6*(6), 1185-1201.

Jiang, N., and Li, L. (2016). Doubly Robust Off-policy Value Evaluation for Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning*, in PMLR 48:652-661.

Konidaris, G., and Osentoski, S. (2008). Value Function Approximation in

Reinforcement Learning using the Fourier Basis. *Computer Science Department Faculty Publication Series*, 101.

Mahmood, A. R., and Sutton, R. S. (2015). Off-policy learning based on weighted importance sampling with linear computational complexity. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*.

Mahmood, A. R., van Hasselt, H., and Sutton, R. S. (2014). Weighted importance sampling for off-policy learning with linear function approximation. *In Advances in Neural Information Processing Systems 27*, Montreal, Canada.

Mahmood, A. R., Yu, H., and Sutton, R. S. (2017). Multi-step off-policy learning without importance sampling ratios. *arXiv:1702.03006*.

Mahmood, A. R., Yu, H., White, M., and Sutton, R. S. (2015). Emphatic temporal-difference learning. In *arXiV:1507.01569*.

Munos, R., Stepleton, T., Haruytunyan, A., and Bellemare, M. G. (2016). Safe and efficient off-policy reinforcement learning. *arXiv:1606.02647*.

Precup, D., Sutton, R. S., Paduraru, C., Koop, A., and Singh, S. (2006). Off-policy learning with recognizers. In *Advances in Neural Information Processing Systems 18*.

Precup, D., Sutton, R. S., and Singh, S. (2000). Eligibility traces for off-policy policy evaluation. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 759-766. Morgan Kaufmann.

Ross, S. M. (2013). *Simulation*. San Diego: Academic Press.

Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo Method*. Wiley, New York..

Rummery, G. A., and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report CUEF/F-INFENG/TR 166, Engineering Department, Cambridge University.

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

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky, D. S. and Hasselmo, M. E. (eds.), *Advances in Neural Information Processing Systems 8*, pp. 1038-1044. MIT Press.

Sutton, R. S., and Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). Manuscript in preparation.

Sutton, R. S., Mahmood, A. R., Precup, D., and van Hasselt, H. (2014). A new Q(λ) with interim forward view and Monte Carlo equivalence. In *Proceedings of the 31st International Conference on Machine Learning*, Beijing, China.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, pp. 761–768, Taipei, Taiwan.

Sutton, R. S., Szepesvari, C., and Maei, H. R. (2009). A convergent O(n) algorithm for off-policy temporal difference learning with linear function approximation. *Advances in Neural Information Processing Systems 21.*

Thomas, P. and Brunskill, E. (2016). Data-Efficient Off-Policy Policy Evaluation for Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning*, in PMLR 48:2139-2148.

van Seijen, H., and Sutton, R. S. (2014). True online TD(λ). In *International Conference on Machine Learning 31.*

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

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* PhD thesis, Cambridge University.

Yu, H., Mahmood, A. R., and Sutton, R. S. (2017). On generalized Bellman equations and temporal difference learning. *arXiv:1704.04463.*