Estimating Variance of Returns using Temporal Difference Methods

by

Brendan Bennett

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) methods provide a powerful means of learning to make predictions in an online, model-free, and highly scalable manner. In the reinforcement learning (RL) framework, we formalize these prediction targets in terms of a (possibly discounted) sum of rewards, called the return. Historically, RL methods have mainly focused on learning to estimate the expected return, or the value, but there has been some indication that using TD methods to make more general predictions would be desirable. In this thesis, we describe an approach to making such predictions, with emphasis on estimating the variance of the return.

Equipped with an estimate of the variance, a learning agent can gauge not just the mean outcome in a given situation, but also the degree to which an individual return will tend to deviate from the average. Such knowledge could be applied towards expressing more sophisticated predictions, decision making under uncertainty, or hyperparameter optimization, among other things.

Previous work has shown that it is possible to construct an approximate Bellman equation for higher moments of the return using estimates of the preceding moments, which can then be used in a TD-style algorithm to learn those moments. This approach builds on the *raw* moments of the return, which tend to make for poor approximation targets due to the outsize effect that noise and other sources of error have on them. In contrast, the *central* moments generally make for more robust approximation targets. Learning to estimate the return's second central moment, *i.e.* the variance, would be useful on its own and as a prelude to future algorithms.

However, defining a suitable prediction target for the return's variance is not straightforward. The expected return is easily expressed as a Bellman equation; variance, as a nonlinear function of the return, is hard to formulate in similar terms. Establishing convergence for nonlinear algorithms is more difficult as well.

Our main contributions concern an algorithm that attempts to navigate these issues: Direct Variance Temporal Difference Learning (DVTD). It consists of two components: the first learns the value function, while the second learns to predict the discounted sum of squared TD errors emitted by the value learner. This δ^2-return is equivalent to the variance of the original return when the value function is unbiased. We provide an analysis demonstrating this equivalence, which also illuminates the relationship between the δ^2-return and alternative various moment-based targets.

For the more typical case where the true value function is unavailable, we provide an interpretation for what DVTD is estimating, and show that it converges to a unique fixed-point under linear function approximation. We also describe how adjusting hyperparameters can yield new approximation targets, allowing us to estimate the variance of the λ-return. Finally, we report on some experiments indicating DVTD's superior performance relative to alternative methods, which also serve to validate our claims regarding DVTD's stability and practical usability.

Gesse at this woorke as happe doth leade.
By chance to truthe you may procede.
And firste woorke by the question,
Although no truthe therein be don.
Suche falsehode is so good a grounde,
That truth by it will soone be founde.

— Robert Recorde, c. 1542

Acknowledgements

I wish to express my sincere gratitude to everyone who has supported me during the long and occasionally trying process of finishing this thesis. I'm happy to say it's finally done!

The paramount influence on my academic endeavours has been my supervisor Rich Sutton. Years ago, I enrolled in his course thinking that it would be somewhat interesting; instead I found it to be life-changing. His mentorship has informed my views on artificial intelligence, but also on what constitutes good research: a resolution to solve the hard problems, dedication to getting the details right, and the eschewing of gratuitous Latin or footnotes[1].

Others from my time at the University of Alberta deserve thanks as well. When I was just starting out, Rupam Mahmood, Joseph Modayil, Harm van Seijen, Janey Yu, and Patrick Pilarski were wonderful mentors, generous with their time and expertise, and could be counted on to provide at least one insight with every conversation. Janey, Touqir Sajed, and Pooria Joulani were particularly helpful when I was learning about modelling stochastic approximation with differential equations.

I am grateful to the friends and colleagues I got to know during my time with the RLAI group.

Craig Sherstan, Dylan Ashley, Kenny Young, Adam White, and Martha White were part of the merry bunch responsible for the initial theory and experiments for DVTD. In addition to being exemplary co-authors, they are great friends as well. Working through ideas with Craig made a lot of things snap into place, and his dedication to extending the frontiers of AI is a continuing source of inspiration. Arguing with Dylan has been one of my favourite sources of recreation and is not infrequently

[1] I hope that two out of three is acceptable, *sub specie aeternitatis*.

educational; Kenny is similarly up for wide-ranging discussion, particularly when we got lost wandering through an unfamiliar city. Adam was very helpful in refining what experiments I should focus on and identifying which questions would yield the most interesting answers. Martha provided advice at a number of junctures, and much of my knowledge of the finer points of RL-related math comes from reading her papers.

Wesley Chung, Muhammad Zaheer, and Vincent Liu provided valuable help clarifying some ideas about more general forms of the return and estimating functions of the return in particular. The discussions (and animated GIFs) shared with Roshan Shariff, Alex Kearney and Kris de Asis were also mind-expanding, although they are only present in this work indirectly.

I would also like to thank my defence committee: Rich, Csaba Szepesvári, and Dale Schuurmans. Despite the lockdown, they were clearly not phoning it in. Even after all the work I've put in, I felt I achieved greater understanding as a result of their questions and comments, and now have numerous further ideas I intend to explore.

Naturally, I would also like to thank my family for their support; I'm honoured and touched by their belief in me. My friends were similarly crucial– their willingness to be used as a sounding board while I tried to get at the heart of the matter suggests they either have saintlike patience or a perverse affinity for listening to fumbling explanations.

Without the aforementioned, I doubt I would have made it to the end of this process.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

When making decisions or planning for the future, the single most valuable piece of information is the *expected value*, that is, what you would expect to happen "on average".

It requires a certain amount of experience to gauge what response to a given scenario yields the best outcome; choosing an action and executing it requires a certain amount of effort as well. Part of the allure of machine learning (ML) is that, at least in some circumstances, this process can be automated, and computers are well-suited to processing vast quantities of data and making decisions quickly. With a little creativity, machine learning can be applied in a surprising number of situations.

Of particular interest are decision problems where the effects of taking an action is only revealed at some point in the future. Many tasks can be modelled in this fashion. For example, when playing a game, each move contributes to victory or defeat, but only on the final move is the outcome made apparent. More abstractly, even composing a sentence fits this model, as each word affects the overall meaning of the thought being conveyed.

Of particular interest are decision problems that can be recursively decomposed into sub-problems, where the result of taking a particular action does not depend on the prior history of the system up to that point. These sorts of tasks which are well-suited to a certain kind of machine learning, called *reinforcement learning* (RL).

The RL framework models systems as consisting of a learning *agent* in some *environment*. At each time step, the agent observes the state of its environment,

executes an action, which results in the environment transitioning to a new state and emitting a reward. If the agent's action selection policy is fixed, our goal might be to learn to predict the (possibly discounted) sum of rewards it can expect to receive conditioned on its observations; alternatively, it might be looking to improve its policy to maximize the rewards it will receive. This sum of rewards is called the *return*, and can be expressed mathematically as

$$\underbrace{G_t}_{\substack{\text{return} \\ \text{from time ``}t\text{''}}} = \sum_{n=1}^{\infty} \overbrace{\gamma^{n-1}}^{\substack{\text{discount factor,} \\ \gamma \in [0,1]}} \underbrace{R_{t+n}}_{\text{transition reward}}.$$

Reinforcement learning is appealing because it is flexible, lending itself to algorithms that can learn incrementally, continuously, not to mention efficiently. Insofar as your task can be expressed in terms of the expected sum of rewards, then a RL algorithm is probably the method of choice for addressing it. Typically, an RL task is formulated with reference to the *expected return*, also called the *value*.

If you know an action's expected outcome, the *next* most important thing to be aware of is that the average case doesn't always tell the whole story.

One can think of many tasks that seem to have a natural reward function, only to find that the "obvious" return does not quite capture the problem's essence. For example, in games with binary outcomes, if a given line of play is ultimately expected to result in a loss, then perhaps a riskier style might be called for, even if it is expected to lose by a wider margin. Conversely, a traveller in an autonomous car might prefer a route that takes longer so long as that route poses no risk of traffic jams which might make them late for an appointment.

This presents a dilemma for RL practitioners, since it can be difficult or even impossible to express these sorts of considerations in the form of a discounted sum. Rather than abandoning the myriad benefits of RL, we might instead seek a means of adapting our methods by making more sophisticated predictions.

The typical solution is usually to modify the task specification in an attempt to have the rewards better represent what we truly wish to predict or optimize while obeying the constraints of the framework. This can have adverse effects, in some

cases making the task more difficult for an agent to learn, or in other cases changing the meaning of success to the extent that the agent learns the wrong thing.

An alternative would be instead learn to estimate *functions* of the return, which would allow us to keep thinking in terms of easily measured quantities (like return on investment, the probability of winning a game, or travel-time to a destination), while enabling us to address more complicated questions of utility as well. In this paradigm, we could learn from the natural prediction target while reserving the ability to adjust it to bring it as necessary.

To give another example, successful gamblers must account for the risk associated with a bet in addition to the expected profit. If the odds are good but the outcome is uncertain, it might be better not to play if there's the possibility of ending up in debt to unsavory characters. We could of course modify how we defined the return in order to penalize losses appropriately, but this reduces the usefulness of the learned predictions. In other circumstances, our agent might not be so risk-averse, and so would have to re-learn its gambling strategy because its past experience is no longer applicable. If, instead, we had learned to estimate the return's distribution, we could instead approximate how worthwhile a particular bet would be under various different circumstances.

As a test case, this thesis focuses on a method for estimating the return's variance, which has historically been ignored in comparison to the expected return. This may be partially due to the difficulty involved with analyzing algorithms that target it using existing methods. The variance of a random variable is its the second central moment, and provides some notion of how individual realizations of that random variable differ from its mean; it can be interpreted as a measure of uncertainty or risk.

The algorithm we analyze, called Direct Variance Temporal Difference Learning (DVTD), is based off the well-studied TD(λ) algorithm, and enjoys many of that celebrated algorithm's advantages. DVTD manages this by first learning an estimate of the expected return, and then uses that estimate to construct a prediction target which approximates the variance of the return.

While we cannot claim this algorithm to be entirely new[1], this thesis does contain some novel contributions:

- We provide a detailed analysis and explication of DVTD, showing how it relates to previous work and comparing it with alternative methods for estimating the return's variance.
- We extend DVTD to the general value function setting, show that it can be used to construct prediction targets for the variance of the λ-return, and even the variance of returns with different bootstrapping hyperparameters.
- We show that DVTD converges under standard conditions, indicating that it can be used in the same situations where TD(λ) would be used.
- Using simulated experiments, we compare DVTD with related methods, including second moment based and distributional algorithms, showing that DVTD generally holds the advantage over the alternatives.
- We also provide some general background for methods of estimating arbitrary moments of the return, and describe how they might be used to approximate arbitrary functions of the return.

In Chapter 2, we provide a (relatively) brief summary of reinforcement learning. Chapter 3 contains some further background information, with some more commentary on functions of the return and why learning to approximate them is a worthwhile endeavor. The remaining chapters contain the bulk of our contributions. In Chapter 4, we introduce an approximation target (the δ^2-return) and an algorithm (DVTD) for estimating the variance of the return. We describe experiments we conducted to test DVTD and compare it with some alternatives in Chapter 5. Finally, we end with a summary of our contributions and some discussion of potential future work in Chapter 6. The remaining pages consist of appendices which provide further details, along some supplemental material such as an algorithm listing and a glossary.

[1] As it was developed in collaboration with others at the University of Alberta's RLAI Lab (Sherstan et al. 2018).

Chapter 2

Background

In this chapter, we provide an overview of reinforcement learning, particularly the parts that we will make reference to in later chapters. We outline the basic framework, discuss temporal difference methods for learning the value function, as well as summarizing how algorithms can be proved convergent using results from the theory of differential equations.

2.1 Reinforcement Learning

Machine learning is about developing algorithms that, once trained, produce an appropriate output for a supplied input. This output usually takes the form of a prediction or action; in traditional supervised learning, the agent's response can be compared to a desired result and used as feedback to improve the agent. However, in many cases, the outcome of a given choice is not immediately apparent.

For example, games such as Chess or Go consist of a sequence of many moves that determine the future state of the board and, ultimately, the result of the game itself. Intuitively, it is clear that with enough experience it becomes possible to predict the effects of different actions over various time horizons, but we require a more precise formalism if we want to mechanize the process.

Reinforcement learning (RL) is such a framework. An RL problem consists of a *learning agent* that interacts with an *environment* according to some *policy*. The agent learns to predict how much of a scalar signal, referred to as the *reward* or the *cumulant*, that it will receive over a period of time by following the policy[1]. The

[1]Strictly speaking, not all reinforcement learning algorithms make explicit predictions about the

environment consists of everything external to the agent, and is the source of the agent's observations about the world as well as the reward.

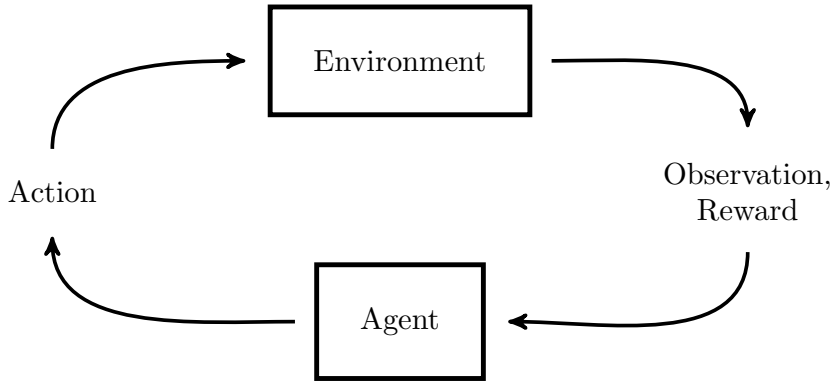

Figure 2.1: *The basic reinforcement learning model. The agent observes the environment, executes an action, causing the environment to emit a reward and provide a new observation.*

In a game of chess, the agent would be the player, and the environment would be the opponent, the board, the legion of cheering spectators, etc. The position of the pieces correspond to the *state*, informs the *actions* taken by the agent; in this example, the actions consist of the moves available to the agent on its turn. The reward would be the outcome of the game, say one point for a win, zero points for a loss, and half a point for a draw, and only emitted after the final move. While the result of the game is not certain until the end of the game, as moves are made it becomes possible to predict the outcome with greater accuracy.

Prediction may either be an end in itself or used as a step towards changing the policy with the goal of maximizing long-term reward. In the *control* case, the agent continually adjusts its policy while simultaneously refining its predictions, looking to maximize the reward it will accumulate. For the *prediction* setting, the policy is held constant and the agent concerns itself only with learning to predict the results of its actions.

value function. For example, policy gradient methods such as REINFORCE(Williams 1992) alter the policy with the goal of maximizing the expected return without actually computing a value for the expected return. In many cases, however, incorporating an estimate of the expected return can improve the performance of these algorithms.

2.2 Formalizing RL using MDPs

We make the ideas introduced in the previous section more concrete by casting them in terms of Markov Decision Process (MDPs).

A Markov Decision Process is a 5-tuple of the form $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where:

- $\mathcal{S}$ represents the "state space", and is a set containing all possible configurations of the environment.
- $\mathcal{A}$, the "action space", is a set containing all actions available to the agent.
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability. We write $P(s, a, s')$ to denote the probability of transitioning to state s' given that action a was taken in state s.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the expected reward function, which gives the average reward associated with a particular transition. In the theory of general value functions, the reward is also referred to as the cumulant.
- γ, which represents discounting, and may be either a constant ($\gamma \in [0, 1]$) or more generally a function of the transition[2], *e.g.* $\gamma : \mathcal{S} \to [0, 1]$.

For ease of analysis we focus on discrete-time MDPs and assume $\mathcal{S}$ and $\mathcal{A}$ are both finite sets.

At each time $t = 0, 1, 2, ...$ the environment's state is S_t, the agent executes an action $A_t \in \mathcal{A}$, and the environment responds by emitting a reward R_{t+1} and transitioning to a state S_{t+1}. The agent selects actions according to some policy (usually denoted π) that assigns a probability to selecting an action conditioned on being in a particular state. This might be denoted $A_t \sim \pi(S_t)$; by convention, we write $\pi(a|s)$ to refer to the probability of selecting action a given state s.

To formalize the notion of long-term cumulative reward, we introduce the *return*, which is the discounted sum of future rewards:

$$G_t \stackrel{\text{def}}{=} \sum_{n=0}^{\infty} \gamma^n R_{t+n+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots, \tag{2.1}$$

[2]Non-constant discounting has some technical advantages at the cost of slightly complicating the notation. For simplicity, we consider state-dependent discounting, meaning that $\gamma_{t+1} = \gamma(S_{t+1})$, although transition dependent discounting (with $\gamma_{t+1} = \gamma(S_t, A_t, S_{t+1})$) behaves similarly). Further details can be found in Sutton, Modayil et al. (2011), Maei (2011), White (2015) and Mahmood (2017), among others.

or, if γ is not constant:

$$G_t \stackrel{\text{def}}{=} \sum_{n=0}^{\infty} R_{t+n+1} \prod_{k=1}^{n-1} \gamma_{t+k} = R_{t+1} + \gamma_{t+1} R_{t+2} + \gamma_{t+1}\gamma_{t+2} R_{t+3} + \cdots. \tag{2.2}$$

It is sometimes more convenient to express the return recursively, as $G_t = R_{t+1} + \gamma_{t+1} G_{t+1}$.

The *value* of a state, written $v_\pi(s)$, is then the expected return, conditional on starting from state s and thereafter selecting actions according to policy π:

$$v_\pi(s) \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{\pi}[G_t | S_t = s] = \mathbb{E}\left[\sum_{n=0} R_{t+n+1} \prod_{k=1}^{n-1} \gamma_{t+k} \middle| S_t = s, A_{t+n} \sim \pi(S_{t+n})\right]. \tag{2.3}$$

For notational purposes, we might elect to denote the variables we condition on as subscripts; for example, the state-value conditioned on π becomes $v_\pi(s) = \mathbb{E}_{\pi,s}[G_t]$.

The return captures the idea of the cumulative reward signal over a period of time, rather than just the immediate reward. Discounting allows us to specify the time horizon that we're interested in, reflecting the notion that the future is uncertain, or that the near future is more important for prediction purposes. The value of a state is simply the average return that an agent will receive starting from that state while following a prescribed policy.

We assumed that P is a function of the current state and action, so $r(s, a, s')$ does not depend on the past history of states and actions; this is the *Markov property*. In other words, knowing the current state provides as much information as knowing the prior states visited, $\mathbb{E}_\pi[G_t | S_0, S_1, \ldots, S_{t-1}, S_t] = \mathbb{E}_\pi[G_t | S_t]$. As such, we can rewrite the value function as:

$$\begin{aligned} v_\pi(s) &= \mathop{\mathbb{E}}_{\pi}[G_t | S_t = s] = \mathop{\mathbb{E}}_{\pi}[R_{t+1} + \gamma_{t+1} G_{t+1} | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} P(s, a, s') \mathop{\mathbb{E}}_{\pi}[R_{t+1} + \gamma_{t+1} G_{t+1} | S_t = s, A_t = a, S_{t+1} = s'] \\ &= \sum_a \pi(a|s) \sum_{s'} P(s, a, s')(r(s, a, s') + \gamma(s') v_\pi(s')). \end{aligned} \tag{2.4}$$

This leads to the following *Bellman equation*:

$$v_\pi(s) = r_\pi(s) + \sum_{s'} P_\pi(s, s') \gamma(s') v_\pi(s'), \tag{2.5}$$

where we use $r_\pi(s) = \mathbb{E}_\pi[R_{t+1}|S_t = s]$ and $P_\pi(s, s') \stackrel{\text{def}}{=} p(S_{t+1} = s'|S_t = s, A_t \sim \pi(S_t))$ to achieve a more compact form.

We can make the preceding much easier to work with by expressing it in terms of linear algebra. Let $\mathbf{P}_\pi$ denote the state-state transition probability matrix such that $[\mathbf{P}_\pi]_{ij} = P_\pi(i, j)$, and let $\mathbf{v}_\pi \in \mathbb{R}$ be the vector whose i-th element is $v_\pi(i)$, the value for state i. Denote by $\mathbf{r}_\pi$ the expected reward vector, with $[\mathbf{r}_\pi]_i = \mathbb{E}_\pi[R_{t+1}|S_t = i]$. Further let $\mathbf{\Gamma}$ be the diagonal matrix whose (i, i)-th entry is the discount factor for state s_i, that is, $[\mathbf{\Gamma}]_{ii} = \gamma(i)$.

Then from (2.5) we have

$$[\mathbf{v}_\pi]_i = [\mathbf{r}_\pi]_i + \sum_j [\mathbf{P}_\pi]_{ij}[\mathbf{\Gamma}]_{jj}[\mathbf{v}_\pi]_j, \tag{2.6}$$

which, after simplification, becomes

$$\mathbf{v}_\pi = \mathbf{r}_\pi + \mathbf{P}_\pi\mathbf{\Gamma}\mathbf{v}_\pi. \tag{2.7}$$

This is a linear functional equation, and can be solved via inversion:

$$\mathbf{v}_\pi = \mathbf{r}_\pi + \mathbf{P}_\pi\mathbf{\Gamma}\mathbf{v}_\pi \quad \Rightarrow \quad (\mathbf{I} - \mathbf{P}_\pi\mathbf{\Gamma})\mathbf{v}_\pi = \mathbf{r}_\pi \tag{2.8}$$

Assuming $\mathbf{I} - \mathbf{P}_\pi\mathbf{\Gamma}$ is invertible, we get

$$\mathbf{v}_\pi = (\mathbf{I} - \mathbf{P}_\pi\mathbf{\Gamma})^{-1}\mathbf{r}_\pi \tag{2.9}$$

Expressions like (2.7) are much easier to work with than those involving multiple nested sums, and allow the solution to be expressed neatly as in (2.9). The only assumption we needed was that $\mathbf{I} - \mathbf{P}_\pi\mathbf{\Gamma}$ be invertible, and this is the case if $\mathbf{P}_\pi$ is an ergodic[3] stochastic[4] matrix and that the discount factor is such that $\gamma(s) < 1$ for at least one state $s \in \mathcal{S}$.

Of course, knowing that a solution exists is not the same as being able to find it or even approximate it. Fortunately, myriad approaches have been developed for

[3]That is, aperiodic and irreducible. A transition matrix is irreducible if each state is (eventually) reachable from every other state. For the matrix to be aperiodic, we must have that for every state there is no integer $n > 1$ such that visits to that state always occur in multiples of n time-steps.

[4]Here meaning a matrix with nonnegative entries such that the sum of each row is one.

solving MDPs. We discuss some of them in the next section.

2.3 Learning the Value Function

If we have access to $\mathbf{P}_\pi$, $\mathbf{\Gamma}$, and $\mathbf{r}_\pi$, then determining the value of a policy is just a matter of matrix inversion. When the transition probabilities or expected rewards are unknown, we might estimate them by recording the transitions and rewards associated with each state.

For example, we could run the policy on the environment and build estimates of $\hat{\mathbf{r}} \approx \mathbf{r}_\pi$ and $\hat{\mathbf{P}} \approx \mathbf{P}_\pi$, then solve for $\hat{\mathbf{v}} = (\mathbf{I} - \hat{\mathbf{P}})^{-1}\hat{\mathbf{r}}$. If we do this naively, the memory and computation requirements can quickly become burdensome for moderately large MDPs; furthermore, the number of samples required to explore the state-space and establish a reasonable estimate for the entries of $\hat{\mathbf{P}}$ and $\hat{\mathbf{r}}$ might be excessive as well.

A more natural approach would be to record trajectories in the MDP, recursively compute the returns, and then average them for each state, as in Monte Carlo Prediction[5]. These algorithms target the complete return (referred to as the *Monte Carlo return*, abbreviated *MC return*), and therefore learns an unbiased estimate of $\mathbf{v}_\pi$.

Various Monte Carlo algorithms are described in Sutton and Barto (2018, Chapter 5) (see pg. 92 in particular). They are examples of a *tabular* method[6], because they essentially build and update a table of state-values. While less onerous than the matrix inversion approach, tabular methods still require at least enough memory to store $N = |\mathcal{S}|$ values; this can still be problematic with a large enough state-space.

[5]Named with reference to the famous casino in Monte Carlo (Metropolis 1987). In general usage, Monte Carlo algorithms approximate a random process by actually simulating it; in RL, however, referring to something as a Monte Carlo method tends to imply that it updates based on the entire return.

[6]Although Sutton and Barto (2018) notes that it's straightforward to extend many of them to function approximation, at least in the on-policy case.

2.4 Function Approximation

As the number of states in our problem grows larger, tabular methods become proportionately less practical. Beyond the memory needed to store estimates for each state, the tabular approach can also require a huge number of samples in order to explore the state space and establish a reliable estimate for the state values. In such cases, we might parameterize the value function and adapt those parameters to approximate $\mathbf{v}_\pi$. Such a value function has the form $f : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$, where $\boldsymbol{\theta} \in \mathbb{R}^d$ is a d-dimensional parameter vector. For concision we might instead write $\mathbf{v}_{\boldsymbol{\theta}}$ instead, *i.e.*,

$$f(s; \boldsymbol{\theta}) = v_{\boldsymbol{\theta}}(s) \approx v_\pi(s). \tag{2.10}$$

Appropriately, this approach is referred to as *function approximation*.

The machine learning literature overflows with possible approximation schemes, but *linear* function approximation (LFA) is perhaps the best explored. In this setting, we map each state to a *feature vector*, for example $\mathbf{x} : \mathcal{S} \to \mathbb{R}^d$. The approximate value for a state is then just the inner product of that $\mathbf{x}(s)$ and $\boldsymbol{\theta}$:

$$v(s) = \boldsymbol{\theta}^\top \mathbf{x}(s). \tag{2.11}$$

Letting $|\mathcal{S}| = N$, we can define a matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$ where each row encodes the feature vector for the corresponding state:

$$\mathbf{X} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{x}(1) \\ \mathbf{x}(2) \\ \vdots \\ \mathbf{x}(N) \end{bmatrix} = \begin{bmatrix} x_1(1) & x_2(1) & \dots & x_d(1) \\ x_1(2) & x_2(2) & \dots & x_d(2) \\ \vdots & & \ddots & \vdots \\ x_1(N) & x_2(N) & \dots & x_d(N) \end{bmatrix}. \tag{2.12}$$

This allows us to express the value function in vector form as $\mathbf{v} = \mathbf{X}\boldsymbol{\theta}$.

Linear function approximation in particular has a number of advantages: the approximations can be simply expressed and usually cheaply computed, and it lends itself to proving stronger convergence or stability results than are usually possible

under more complicated function approximation schemes[7]. From a practical perspective, LFA is generally worth trying first because it is easy to implement and its failure or success can reveal more about the underlying structure of the problem.

2.4.1 Objectives Under Function Approximation

For problems where function approximation is actually necessary, the approximate value function will never be exactly equal to the true value function. How, then, can we determine which weights yield the best approximation? Typically, we do this by defining an *objective* or *error function*, which measures the distance between our approximation and its target.

For reinforcement learning in particular, the standard objective is the *Mean Squared Value Error*, denoted $\overline{\text{VE}}$, defined in Sutton and Barto (2018, pg. 199) as:

$$\overline{\text{VE}}(\mathbf{v}) \stackrel{\text{def}}{=} \|\mathbf{v}_\pi - \mathbf{v}\|_\mu^2 = \sum_{s \in \mathcal{S}} \mu(s)[v_\pi(s) - v(s)]^2. \tag{2.13}$$

$\overline{\text{VE}}$ is the Euclidean distance between $\mathbf{v}$ and $\mathbf{v}_\pi$ weighted according to some distribution $\mu : \mathcal{S} \to \mathbb{R}$ with $\mu(s) \geq 0 \; \forall s \in \mathcal{S}$ and $\sum_{s \in \mathcal{S}} \mu(s) = 1$.

For on-policy prediction, we usually set μ to the *on-policy distribution*, denoted d_π, which weights states proportional to how often the agent visits them. In continuing tasks, this is just the stationary distribution[8].

One advantage of using $\overline{\text{VE}}$ as our objective is that minimizing it corresponds to solving a weighted least squares regression problem:

$$\boldsymbol{\theta}_{\text{LS}} \stackrel{\text{def}}{=} \operatorname*{argmin}_{\boldsymbol{\theta}} \left\|\hat{\mathbf{X}}\boldsymbol{\theta} - \mathbf{v}_\pi\right\|_\mu^2 = (\mathbf{X}^\top \mathbf{D}_\mu \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{D}_\mu \mathbf{v}_\pi, \tag{2.14}$$

where $\mathbf{D}_\mu \stackrel{\text{def}}{=} \mathsf{diag}(\mu(1), \mu(2), \ldots, \mu(N))$.

The associated value function, $\mathbf{v}_{\text{LS}} = \mathbf{X}\boldsymbol{\theta}_{\text{LS}}$, is in some sense the closest we can get to the $\mathbf{v}_\pi$ under linear function approximation. There are a number of algorithms

[7] More general function approximation (for example, neural networks) may be able to represent more complicated functions, but can take longer to train, and may converge to different solutions depending on the details of the training process, if they converge at all.

[8] That is, $\mathbf{d}_\pi = \mathbf{d}_\pi \mathbf{P}_\pi$. If $\mathbf{P}_\pi$ is ergodic, then $\mathbf{d}_\pi$ is the unique left eigenvector with modulus one. For the definition of the on-policy distribution in the episodic case, see Sutton and Barto 2018, pg. 199.

whose estimates converge to $\boldsymbol{\theta}_{\text{LS}}$; for example, the already mentioned Monte Carlo Prediction.

In practice, however, methods targeting the Monte Carlo return can still perform poorly despite being unbiased. The Monte Carlo return, being comprised of a sum of random variables, can exhibit high variance, requiring a huge number of samples to get an accurate estimate. Furthermore, Monte Carlo methods require many transitions to perform an update[9], which can limit their efficacy in continuing tasks or those with long episodes.

2.5 Temporal Difference Learning

The recursive nature of the Bellman equation allows us to avoid having to record the full sequence of states and rewards when learning the value function. For a transition of the form (S_t, R_{t+1}, S_{t+1}), we define the *temporal difference error*, denoted δ_t, as:

$$\delta_t \stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1} v_t(S_{t+1}) - v_t(S_t), \tag{2.15}$$

where v_t is the value function at time t.

If our value function is correct, we should have $\mathbb{E}_\pi[\delta_t] = \mathbb{E}_\pi[R_{t+1} + \gamma_{t+1} v_t(S_{t+1}) - v_t(S_t)] = 0$ by definition.

Temporal difference (TD) learning (Sutton 1988) applies this observation by using δ_t to define an update rule, yielding a learning agent that can learn incrementally without needing the full return. TD methods define approximation targets by substituting their estimate for value function for the full return, *bootstrapping*[10] from their own initial value function.

The simplest TD method is tabular TD(0), which can be expressed quite suc-

[9] There exist online and incremental Monte Carlo algorithms, but we note that for an update to have truly concluded they still require the trajectory to terminate; otherwise they are effectively truncating the return.

[10] In the sense of the colloquial expression "to pull yourself up by your own bootstraps". The agent starts in state of ignorance, but gradually learns about the environment, using its own initially incorrect estimates to improve its approximation of the value function, until eventually it attains an accurate understanding of the world it is embedded within.

cinctly:

$$v_{t+1}(S_t) = v_t(S_t) + \alpha_t \delta_t,$$

where α is the learning rate or stepsize parameter, and δ_t is the *temporal-difference error*. As the agent visits each state, it improves its estimate for that state's value, gradually building a table of state-values consistent with the Bellman equation.

Each individual update is defined in reference to a single transition, in contrast to Monte Carlo methods whose updates can only be said to be complete when the trajectory has terminated. This can lead to substantial performance gains from the reduced variance, even though the fixed-point of Tabular TD(0) is still the true value function v_π.

2.5.1 TD(0) with Linear Function Approximation

Temporal difference learning can be extended to linear function approximation rather easily. At time t, let $\mathbf{x}_t$ be the feature vector, and $\boldsymbol{\theta}_t$ be the weight vector. Then the TD(0) update equations are:

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma_{t+1} \overbrace{\boldsymbol{\theta}_t^\top \mathbf{x}_{t+1}}^{v_t(S_{t+1})} - \overbrace{\boldsymbol{\theta}_t^\top \mathbf{x}_t}^{v_t(S_t)} \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_t \delta_t \mathbf{x}_t. \end{aligned} \tag{2.16}$$

The temporal difference error is unchanged, but instead of updating a single entry in the table of state-values, we update each entry in the weight vector proportional to the corresponding entry in the feature vector. We retain the advantages of tabular TD(0), only needing information from the transition $(S_t, A_t, R_{t+1}, S_{t+1})$ to perform an update, with a per time-step computational cost of $\mathcal{O}(d)$, where d is the dimensionality of the weight vector.

Under the usual conditions, on-policy linear TD(0) is stable and convergent, with a unique fixed point, which we denote $\boldsymbol{\theta}_{\text{TD}(0)}$. We can solve for its fixed-point via[11]:

$$\boldsymbol{\theta}_{\text{TD}(0)} = (\mathbf{X}^\top \mathbf{D}_\pi (\mathbf{I} - \mathbf{P}_\pi \boldsymbol{\Gamma}) \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{D}_\pi \mathbf{r}_\pi. \tag{2.17}$$

[11]For details on how these fixed-point equations are derived, see Tsitsiklis and Van Roy (1997), Sutton, Mahmood and White (2015) and Mahmood (2017).

The TD(0) fixed point (2.17) is similar to that of the least-squares solution (2.14), but under function approximation the two are generally different. The bias introduced by bootstrapping can lead to higher $\overline{\text{VE}}$, but TD(0)'s lower variance can mean better performance in practice.

2.5.2 TD(λ)

The TD(λ) algorithm (Sutton 1989) generalizes TD(0) using *eligibility traces*. The idea is to keep a *trace* of the features activated during each time-step, updating the weights according to the trace rather than just the current features.

Adjusting the bootstrapping hyperparameter λ allows the agent to vary how much it relies on its own value estimates versus the reward sequence. When $\lambda = 0$, the updates are the same as in TD(0) (*c.f.* (2.16)), while for $\lambda = 1$, the traces are a discounted record of the feature activations, making TD(1) essentially a form of online Monte Carlo[12]. For values of λ between 0 and 1, we can think of TD(λ) as interpolating between the TD(0) solution and the MC solution.

The update equations for TD(λ) with accumulating traces are barely more complicated compared to those for TD(0):

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma_{t+1}\boldsymbol{\theta}_t^\top \mathbf{x}_{t+1} - \boldsymbol{\theta}_t^\top \mathbf{x}_t \\ \mathbf{z}_t &= \gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}_t \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_t \delta_t \mathbf{z}_t. \end{aligned} \tag{2.18}$$

It turns out that updating with eligibility traces actually leads to a new approximation target[13], called the *λ-return* and defined by:

Definition 2.1 (λ-return)

For some value function $v : \mathcal{S} \to \mathbb{R}$, we denote by G_t^λ the *λ-return*, denoted G_t^λ and

[12] We can think of TD(1) as a form of every-visit Monte Carlo under exponential averaging determined by α.

[13] See Sutton and Barto (2018, Chapter 12, pg. 287-) for further details.

defined as:

$$\begin{aligned} G_t^\lambda &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}(1-\lambda_{t+1})v(S_{t+1}) + \gamma_{t+1}\lambda_{t+1}G_{t+1}^\lambda \\ &= \sum_{n=1}^{\infty}\left(\prod_{k=1}^{n-1}\gamma_{t+k}\lambda_{t+k}\right)(R_{t+n} + \gamma_{t+n}(1-\lambda_{t+n})v(S_{t+n})), \end{aligned} \tag{2.19}$$

where we use the convention that $\prod_i^j a_i = 1$ for $i > j$. For constant γ and λ, it simplifies to

$$G_t^\lambda = \sum_{n=1}(\gamma\lambda)^{n-1}(R_{t+n} + \gamma(1-\lambda)v(S_{t+n}). \tag{2.20}$$

Different values of λ induce a different approximation target, which in turn lead to a different fixed-point for the weights[14]:

$$\begin{aligned} \boldsymbol{\theta}_{\text{TD}(\lambda)} &= \left[\mathbf{X}^\top\mathbf{D}_\pi(\mathbf{I} - \mathbf{P}_\pi\boldsymbol{\Gamma}\boldsymbol{\Lambda})^{-1}(\mathbf{I} - \mathbf{P}_\pi\boldsymbol{\Gamma})\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{D}_\pi(\mathbf{I} - \mathbf{P}_\pi\boldsymbol{\Gamma}\boldsymbol{\Lambda})^{-1}\mathbf{r}_\pi \\ &= \mathbf{A}_\lambda^{-1}\mathbf{b}_\lambda, \end{aligned} \tag{2.21}$$

where

$$\begin{aligned} \mathbf{A}_\lambda &\stackrel{\text{def}}{=} \mathbf{X}^\top\mathbf{D}_\pi(\mathbf{I} - \mathbf{P}_\pi\boldsymbol{\Gamma}\boldsymbol{\Lambda})^{-1}(\mathbf{I} - \mathbf{P}_\pi\boldsymbol{\Gamma})\mathbf{X}, \\ \mathbf{b}_\lambda &\stackrel{\text{def}}{=} \mathbf{X}^\top\mathbf{D}_\pi(\mathbf{I} - \mathbf{P}_\pi\boldsymbol{\Gamma}\boldsymbol{\Lambda})^{-1}\mathbf{r}_\pi. \end{aligned} \tag{2.22}$$

Values of λ less than one lead to a biased solution, but the updates tend to have lower variance. Empirically, some amount of bootstrapping usually speeds learning. For constant λ with linear function approximation, TD(λ) will eventually converge (Tsitsiklis and Van Roy 1997; Mahmood 2017), but the value that works best can be highly problem dependent.

2.6 Differential Equations and Stochastic Approximation

Proving that a learning algorithm behaves reliably, in particular that it is stable and convergent, can be done in a variety of ways, but one of the most powerful techniques invokes concepts from the theory of differential equations.

We start by observing that many stochastic approximation schemes have the

[14]See Sutton, Mahmood and White (2015) for a very readable derivation of this result, as well as Mahmood's thesis (Mahmood 2017) for a detailed examination of general value functions, including a surprising result on the instability of on-policy TD(λ) with state-dependent bootstrapping.

general form:

$$\mathbf{z}_{n+1} = \mathbf{z}_n + \alpha_n \mathbf{Y}_{n+1}, \tag{2.23}$$

for some sequence of d-dimensional random variables $\{\mathbf{Y}_n\}$.

Each $\mathbf{Y}_n$ might represent *e.g.* stochastic gradients, but could also stand for some more exotic means of updating the parameter vector $\mathbf{z}$. Of particular interest is when $\{\mathbf{Y}_n\}$ is such that there is some continuous function $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^d$ such that

$$\mathbf{f}(\mathbf{z}_{n+1}) = \mathbb{E}[\mathbf{Y}_{n+1} | \mathbf{Y}_0, \mathbf{Y}_1, \ldots, \mathbf{Y}_n, \mathbf{z}_0], \tag{2.24}$$

allowing us to write (2.23) as:

$$\mathbf{z}_{n+1} = \alpha_n [\mathbf{f}(\mathbf{z}_n) + \mathbf{M}_{n+1}]. \tag{2.25}$$

Here:

- $\{\alpha_n\}$ denotes a sequence of real-valued stepsizes.
- $f : \mathbb{R}^d \to \mathbb{R}^d$ denotes a deterministic map
- $\{\mathbf{M}_{n+1}\} = \mathbf{Y}_{n+1} - \mathbf{f}(\mathbf{z}_n)$ is a d-dimensional noise sequence.

We can interpret (2.25) as a noisy Euler scheme[15] for the following differential equation:

$$\dot{\mathbf{z}}(t) = \mathbf{f}(\mathbf{z}(t)). \tag{2.26}$$

The evolution of (2.26) only corresponds to the iterates of (2.23) when there is no noise[16] and the stepsize approaches zero. At first glance, this might seem excessively restrictive. However, under the right conditions, the two systems are similar enough that we can learn about the behavior of the SA algorithm by analyzing the limiting ODE, which is fortunate because deriving results about continuous systems can be much easier.

To permit comparison between the SA method and its limiting ODE at arbitrary

[15] A discretization of a continuous ordinary differential equation. In general, Euler schemes are numerical methods for solving ODEs, see Press et al. (2007, Chapter 17, pg. 907) or a similar work on numerical computing for further details.

[16] That is, $\mathbf{M}_{n+1} = \mathbf{0}\ \forall n \geq 0$

points in time, we define the *interpolated ODE*. Let t_n be defined as

$$t_{n+1} \stackrel{\text{def}}{=} t_n + \alpha_n = \sum_{k=0}^{n} \alpha_k, \quad \text{with } t_0 \stackrel{\text{def}}{=} 0, \tag{2.27}$$

then the interpolated ODE, denoted $\overline{\mathbf{z}}(t)$, is:

$$\overline{\mathbf{z}}(t) \stackrel{\text{def}}{=} \overline{\mathbf{z}}(t_n) + \left(\frac{t - t_n}{\alpha_n}\right)[\overline{\mathbf{z}}(t_{n+1}) - \overline{\mathbf{z}}(t_n)]. \tag{2.28}$$

Then $\overline{\mathbf{z}}(t)$ has the value of $\mathbf{z}_n$ for $\overline{\mathbf{z}}(t_n)$, and for $t \in (t_n, t_{n+1})$ is the convex combination of $\mathbf{z}_n$ and $\mathbf{z}_{n+1}$. If we can establish that $\overline{\mathbf{z}}(t)$ remains "close" to the limiting ODE, then proving results about $\mathbf{z}(t)$ provides a means of establishing a similar claim for the associated SA scheme.

2.6.1 The Limiting ODE of TD(0)

To give a concrete example, we examine TD(0) in the ODE framework. For convenience, we assume that the rewards depend only on the state-state transition[17]:

Assumption 2.1 (Transition Rewards)

The rewards for a given transition is independent of the action taken:

$$r(s, a, s') = r(s, s') \stackrel{\text{def}}{=} \mathbb{E}_\pi[R_{t+1} | S_t = s, S_{t+1} = s']. \tag{2.29}$$

With $\mathcal{A}$. 2.1 in mind, we define the expected reward matrix and vector as

$$\begin{aligned} \mathbf{R} &\stackrel{\text{def}}{=} \big[r(i,j)\big] \in \mathbb{R}^{N \times N} \qquad \text{for } i, j = 1, 2, \ldots, N, \\ \mathbf{r} &\stackrel{\text{def}}{=} \big[r(i)\big] \in \mathbb{R}^{N} \qquad\quad \text{for } i = 1, 2, \ldots, N. \end{aligned} \tag{2.30}$$

Furthermore, we assume that the transitions are sampled independently from an identical distribution:

Assumption 2.2 (Independent transition samples)

The sequence $\{(s_n, r_n, s'_n)\}_{n=1}^{\infty}$ are independent and identically distributed samples of (s, r, s').

[17] $\mathcal{A}$. 2.1 serves to simplify the notation and consequently the proofs, but does not really affect the conclusions we can draw. Similar results can be proved when the reward is action dependent, but they are more complicated to express because we must invoke tensor notation to describe them.

$\mathcal{A}$. 2.2 is commonly employed in the literature[18], although it is somewhat unrealistic. As a consequence, $\mathbf{x}(s_n) = \mathbf{x}_n$ and other quantities that depend only on the current transition are also i.i.d.

Then for a transition (s_n, r_n, s'_n), the TD(0) updates have the same form as (2.25):

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \alpha_n[r_n + \boldsymbol{\theta}_n^\top(\gamma'_n\mathbf{x}'_n - \mathbf{x}_n)]\mathbf{x}_n \tag{2.32}$$

$$= \boldsymbol{\theta}_n + \alpha_n[\mathbf{g}(\boldsymbol{\theta}_n) + \mathbf{M}_{n+1}], \tag{2.33}$$

where $\mathbf{g} : \mathbb{R}^d \to \mathbb{R}^d$ is the "mean" or "expected" update, and represents the change in the weights we would observe on average sampling over (s, r, s'). It is defined by

$$\mathbf{g}(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \mathbf{b} - \mathbf{A}\boldsymbol{\theta} = -\mathbf{A}[\boldsymbol{\theta} - \boldsymbol{\theta}^*] \tag{2.34}$$

where $\boldsymbol{\theta}^* = \mathbf{A}^{-1}\mathbf{b}$, with the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ given by[19]:

$$\mathbf{A} = \mathbf{X}^\top\mathbf{D}_\pi(\mathbf{I} - \mathbf{P}_\pi\boldsymbol{\Gamma})\mathbf{X} \qquad \mathbf{A} \in \mathbb{R}^{d\times d}, \tag{2.35}$$

$$\mathbf{b} = \mathbf{X}^\top\mathbf{D}_\pi\mathbf{r}_\pi \qquad \mathbf{b} \in \mathbb{R}^d. \tag{2.36}$$

The noise is denoted $\mathbf{M}_{n+1}$ and is defined as:

$$\mathbf{M}_{n+1} \stackrel{\text{def}}{=} \delta_n(\boldsymbol{\theta}_n)\mathbf{x}_n - \mathbf{g}(\boldsymbol{\theta}_n) = [r_n + \boldsymbol{\theta}_n^\top(\gamma'_n\mathbf{x}'_n - \mathbf{x}_n)]\mathbf{x}_n - [\mathbf{b} - \mathbf{A}\boldsymbol{\theta}_n], \tag{2.37}$$

[18] As others have noted (Dalal et al. 2017, footnote 1), this assumption does not hold in practice; however its use is standard in establishing reinforcement learning results. Other conditions that might replace it are not necessarily more realistic, *e.g.* Tsitsiklis and Van Roy (1997) requires exponential mixing for the Markov chain. For an aperiodic, irreducible Markov chain, the mixing time is indeed exponential (Levin and Peres 2017, Theorem 4.9) in the sense that the distance between the distribution after k time-steps and the stationary distribution $\mathbf{d}_\pi$ shrinks as

$$\left\|\mathbf{s}_0^\top\mathbf{P}_\pi^k - \mathbf{d}_\pi\right\|_1 \leq C\xi^k \tag{2.31}$$

for some $C > 0$ and $\xi \in (0, 1)$ with arbitrary initial distribution $\mathbf{s}_0$. For ξ close to zero, mixing is rapid enough as to make the samples effectively i.i.d., but for ξ close to one the samples are highly correlated.

However, simulations tend to agree with the results derived under the i.i.d. assumption, suggesting that a more careful analysis might succeed in removing it. Intriguingly, such an analysis might also provide indications for convergence acceleration methods if it reveals situations under which the i.i.d. assumption falls apart.

Even if that proves intractable, modifications to the algorithms could unify implementation with theory. For example, in online learning with deep networks, it is common to use a replay buffer with randomized sampling. This reduces correlation between individual samples; transformations applied to the features (Ghiassian et al. 2018) can also help in this regard.

[19] In essence, $\mathbf{A} = \mathbb{E}_\pi[\mathbf{x}_n(\gamma'_n\mathbf{x}'_n - \mathbf{x}_n)^\top]$ and $\mathbf{b} = \mathbb{E}_\pi[\mathbf{x}_n r_n]$, which can be found by rearranging (2.32) and taking the expected value.

where

$$\delta_n(\boldsymbol{\theta}) \stackrel{\text{def}}{=} r_n + \boldsymbol{\theta}^\top(\gamma'_n \mathbf{x}'_n - \mathbf{x}_n) \tag{2.38}$$

is the TD error with weights $\boldsymbol{\theta}$ and a sample (s_n, r_n, s'_n).

Then the limiting ODE of (2.34) is:

$$\boxed{\dot{\boldsymbol{\theta}}(t) = \mathbf{g}(\boldsymbol{\theta}(t)) = \mathbf{b} - \mathbf{A}\boldsymbol{\theta}(t) = -\mathbf{A}(\boldsymbol{\theta}(t) - \boldsymbol{\theta}^*).} \tag{2.39}$$

Equation (2.39) has a closed form solution if $\mathbf{A}$ is invertible[20]:

$$\begin{aligned}
\boldsymbol{\theta}(t) &= e^{-t\mathbf{A}}\left(\boldsymbol{\theta}_0 + \int_0^t e^{\tau\mathbf{A}}\mathbf{b}\,\mathrm{d}\tau\right) \\
&= e^{-t\mathbf{A}}\boldsymbol{\theta}_0 + e^{-t\mathbf{A}}\mathbf{A}^{-1}(e^{t\mathbf{A}} - \mathbf{I})\mathbf{b} \\
&= \mathbf{A}^{-1}\mathbf{b} + e^{-t\mathbf{A}}(\boldsymbol{\theta}_0 - \mathbf{A}^{-1}\mathbf{b}) \\
&= \boldsymbol{\theta}_0 e^{-t\mathbf{A}} + \mathbf{A}^{-1}\mathbf{b}(\mathbf{I} - e^{-t\mathbf{A}}),
\end{aligned} \tag{2.40}$$

where $\boldsymbol{\theta}_0$ is the initial value, *i.e.* $\boldsymbol{\theta}(0) = \boldsymbol{\theta}_0$.

In fact it can be shown that $\mathbf{A}$ *is* invertible (*c.f.* Bertsekas 2012, Chapter 6.3), and furthermore that it is positive definite[21]. That is, the eigenvalues of $\mathbf{A}$ have positive real part, which implies that $\lim_{t\to\infty} e^{-\mathbf{A}(t-s)} = \mathbb{0}$. Therefore:

$$\begin{aligned}
\lim_{t\to\infty} \boldsymbol{\theta}(t) &= \lim_{t\to\infty} \left(\boldsymbol{\theta}_0 e^{-t\mathbf{A}} + \mathbf{A}^{-1}\mathbf{b}(\mathbf{I} - e^{-t\mathbf{A}})\right) \\
&= \cancelto{\mathbb{0}}{\boldsymbol{\theta}_0 e^{-t\mathbf{A}}} + \mathbf{A}^{-1}\mathbf{b} - \cancelto{\mathbb{0}}{e^{-t\mathbf{A}}} \\
&= \mathbf{A}^{-1}\mathbf{b} = \boldsymbol{\theta}^*,
\end{aligned} \tag{2.41}$$

as might be expected.

Recognizing $\mathbf{A}^{-1}\mathbf{b} = \boldsymbol{\theta}^*$ as the TD fixed point, (2.40) can be rewritten as:

$$\boxed{\boldsymbol{\theta}(t, s, \boldsymbol{\theta}_s) = \boldsymbol{\theta}^* + e^{-\mathbf{A}(t-s)}(\boldsymbol{\theta}_s - \boldsymbol{\theta}^*)} \tag{2.42}$$

where we use $\boldsymbol{\theta}(t, s, \boldsymbol{\theta}_s)$ to denote the solution of (2.39) at time t starting from time s with initial value $\boldsymbol{\theta}_s$. Written like this, it is obvious that TD(0)'s limiting ODE (2.42) converges to $\boldsymbol{\theta}^*$ as time goes to infinity.

[20] See Hirsch, Smale and Devaney (2013, Chapter 6, pg. 132.).
[21] In the sense that $\mathbf{z}^\top\mathbf{A}\mathbf{z} > 0$ for $\mathbf{z} \in \mathbb{R}^d$ and $\mathbf{z} \neq \mathbf{0}$.

While it is reassuring that the limiting ODE (2.39) has the same solution as the SA scheme (2.33), we need to verify that the ODE approach accurately captures the behavior of the discrete time updates. Under some reasonable assumptions, this is indeed the case, and eventually the sequence $\{\boldsymbol{\theta}_n\}$ generated by (2.33) becomes arbitrarily close to the trajectory of (2.42).

We delay stating the relevant results, however, because we are interested in the convergence of a broader class of algorithms, which will be described in the following section.

2.6.2 Two Timescale Convergence

Convergence analyses for algorithms like TD(0) benefit from the fact that the expected update (and therefore the limiting ODE) have some "nice" form. In Section 2.6.1, we noted that $\mathbf{A}$ is invertible, so $\dot{\boldsymbol{\theta}} = -\mathbf{A}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)$ has a closed form solution, and since $\mathbf{A}$ is positive definite we could show that asymptotically $\boldsymbol{\theta}(t) \to \boldsymbol{\theta}^*$.

Sadly, not every algorithm has such a convenient form. In particular, there are those with more complicated update methods, where some parameters are updated at different rates or with separate rules for some sub-components.

Fortunately, we can take advantage of existing results on "two-timescale" stochastic approximation schemes. These results hold for algorithms with update rules of the form:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \beta_n[\mathbf{h}(\mathbf{w}_n, \boldsymbol{\theta}_n) + \mathbf{N}_{n+1}], \tag{2.43}$$

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \alpha_n[\mathbf{g}(\mathbf{w}_n, \boldsymbol{\theta}_n) + \mathbf{M}_{n+1}], \tag{2.44}$$

where $\mathbf{w} \in \mathbb{R}^{d_1}$, $\boldsymbol{\theta} \in \mathbb{R}^{d_2}$, β_n and α_n are stepsizes, $\mathbf{h} : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$ and $\mathbf{g} : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \to \mathbb{R}^{d_2}$ are deterministic functions, and $\mathbf{N}_{n+1} \in \mathbb{R}^{d_1}$ and $\mathbf{M}_{n+1} \in \mathbb{R}^{d_1}$ are noise sequences.

The key to establishing convergence for such schemes is to treat one of the components as acting on a "faster" timescale relative to the other[22]. With sufficiently

[22]In the parlance of differential equations, this is part of the theory of "singularly perturbed ODEs".

separated timescales, the slow iterates appears almost constant to the fast component; from the perspective of the slow component, the fast part seems to be a rapidly equilibrated transient[23].

For the purposes of analysis, we can regard

$$\dot{\mathbf{w}}(t) = \mathbf{h}(\mathbf{w}(t), \boldsymbol{\theta}) \tag{2.45}$$

as representing the fast component, with $\boldsymbol{\theta}$ held fixed. If we assume that (2.45) has an asymptotically stable equilibrium as a function of $\boldsymbol{\theta}$ (say $\mathbf{w}^*(\boldsymbol{\theta})$), then we would expect the slow ODE to behave like:

$$\dot{\boldsymbol{\theta}}(t) = \mathbf{g}(\mathbf{w}^*(\boldsymbol{\theta}(t)), \boldsymbol{\theta}(t)). \tag{2.46}$$

This can be interpreted as saying that, on the slower timescale, the fast part is effectively a function of the slow component because it reaches equilibrium (provided one exists) with such alacrity.

There is a substantial amount of previous work on two-timescale systems, but for our purposes the results from Borkar (2008) will suffice. We begin by stating the needed assumptions:

Assumption 2.3 (Stepsize Sequence)

The stepsize sequences $\{\alpha_n\}$ and $\{\beta_n\}$ satisfy

$$\sum_{k=0}^{\infty} \beta_k = \sum_{k=0}^{\infty} \alpha_k = \infty, \qquad \sum_{k=0}^{\infty} \beta_k^2 + \alpha_k^2 < \infty, \qquad \lim_{n\to\infty} \frac{\alpha_n}{\beta_n} = 0, \tag{2.47}$$

with $\beta_k, \alpha_k > 0$ for $k \in \mathbb{N}$.

Which is to say $\{\beta_k\}$ and $\{\alpha_k\}$ are sequences of positive scalars whose sum has no finite limit, but the sum of each sequence's squares is finite.

This assumption is similar to the standard Robbins-Munro conditions, but the additional restriction on the ratio of the stepsizes leads to timescale separation.

[23] An analogy might be drawn to algorithms with a nested loop structure, where after each step in the outer loop, the inner loop is repeated until its updates converge (or at least are smaller than some tolerance).

Hearkening back to (2.27), if we regard time elapsed for the slow component as $T_\alpha(n) = \sum_{i=0}^{n} \alpha_i$, and $T_\beta(n) = \sum_{i=0}^{n} \beta_i$ for the fast component, then $\boldsymbol{\mathcal{A}}$. 2.3 ensures that $T_\alpha(n)/T_\beta(n) \to 0$. We can interpret this as saying the slow component runs for a shorter time than the fast component, given the same number of iterations.

The next assumption ensures that the limiting ODE has a unique solution:

Assumption 2.4 (Lipschitz Mapping)

The maps $\mathbf{h} : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$ and $\mathbf{g} : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$ are Lipschitz[24].

The following conditions ensure that the martingale noise cannot cause divergence:

Assumption 2.5 (Martingale Difference Sequence)

The sequences $\{\mathbf{N}_n\}$ and $\{\mathbf{M}_n\}$ are a square-integrable martingale difference sequences with respect to the family of filtrations

$$\mathcal{F}_n = \sigma(\mathbf{w}_0, \boldsymbol{\theta}_0, \mathbf{N}_1, \mathbf{M}_1, \dots, \mathbf{N}_n, \mathbf{M}_n) = \sigma(\mathbf{w}_0, \boldsymbol{\theta}_0, \{(\mathbf{N}_k, \mathbf{M}_k) : k \leq n\}). \tag{2.49}$$

Assumption 2.6 (Martingale Bound)

For all $n \in \mathbb{N}$ there exists some $K_m > 0$ such that

$$\begin{aligned} \mathbb{E}[\|\mathbf{N}_{n+1}\|^2 | \mathcal{F}_n] &\leq K_m[1 + \|\mathbf{w}_n\|^2 + \|\boldsymbol{\theta}_n\|^2], \\ \mathbb{E}[\|\mathbf{M}_{n+1}\|^2 | \mathcal{F}_n] &\leq K_m[1 + \|\mathbf{w}_n\|^2 + \|\boldsymbol{\theta}_n\|^2], \end{aligned} \tag{2.50}$$

which is to say the martingale difference sequences $\{\mathbf{N}_n\}$ and $\{\mathbf{M}_n\}$ are bounded from above in terms of the norms of the iterates.

Assumption 2.7 (Bounded Iterates)

The iterates of (2.43) and (2.44) remain bounded almost surely:

$$\sup_n \left(\|\mathbf{w}_n\| + \|\boldsymbol{\theta}_n\|\right) < \infty \quad \text{a.s.} \tag{2.51}$$

We also need that each of (2.45) and (2.46) converges on their respective timescales:

[24]Recall that a function is Lipschitz if there exists some positive scalar C such that for all $\mathbf{y}, \mathbf{z} \in \mathbb{R}^d$:

$$\|\mathbf{g}(\mathbf{y}) - \mathbf{g}(\mathbf{z})\| \leq C\|\mathbf{y} - \mathbf{z}\|. \tag{2.48}$$

Assumption 2.8 (Fast Component Convergence)

The fast component,

$$\dot{\mathbf{w}}(t) = \mathbf{h}(\mathbf{w}(t), \boldsymbol{\theta}), \tag{2.45}$$

has a globally asymptotically stable equilibrium that is a function of $\boldsymbol{\theta}$, which we denote by $\mathbf{w}^*(\boldsymbol{\theta})$. Furthermore, $\mathbf{w}^* : \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$ is Lipschitz continuous.

Assumption 2.9 (Slow Component Convergence)

The slow timescale ODE,

$$\dot{\boldsymbol{\theta}}(t) = \mathbf{g}(\mathbf{w}^*(\boldsymbol{\theta}(t)), \boldsymbol{\theta}(t)), \tag{2.46}$$

has a globally asymptotically stable equilibrium $\boldsymbol{\theta}^*$.

Provided these conditions hold, then we can apply some results from Borkar (2008) regarding the system's convergence[25]:

Lemma 2.1 (Fast Timescale Convergence)

Assume $\mathcal{A}$. 2.1 to 2.9. Then the iterates of (2.43) and (2.44) converge almost surely to the internally chain transitive invariant sets of the ODE $\dot{\mathbf{w}}(\mathbf{w}(t), \boldsymbol{\theta}(t))$, $\dot{\boldsymbol{\theta}}(t) = 0$. That is, $(\mathbf{w}_n, \boldsymbol{\theta}_n) \to \{(\mathbf{w}^*(\boldsymbol{\theta}), \boldsymbol{\theta}) : \boldsymbol{\theta} \in \mathbb{R}^{d_2}\}$.

This lemma essentially states that $\mathbf{w}_n$ will asymptotically track $\mathbf{w}^*(\boldsymbol{\theta}_n)$ with probability one, and leads to the main result we will use:

Theorem 2.2 (Overall Convergence)

Assume $\mathcal{A}$. 2.1 to 2.9.

Then the iterates of (2.43) and (2.44) converge to $(\mathbf{w}^*(\boldsymbol{\theta}^*), \boldsymbol{\theta}^*)$ almost surely. More precisely:

$$\Pr\left(\lim_{n\to\infty} (\mathbf{w}_n, \boldsymbol{\theta}_n) = (\mathbf{w}^*, \boldsymbol{\theta}^*)\right) = 1 \tag{2.52}$$

which states that the iterates of the stochastic approximation scheme will ultimately converge to the fixed point of the two-timescale ODE.

[25] See Chapter 6 of Borkar 2008, in particular Lemma 1 and Theorem 2.

Chapter 3

Functions of the Return

In this chapter, we discuss learning functions of the return as an alternative to modifying the return to make a problem more amenable to reinforcement learning. We describe some of the ways this formulation can make task specifications more precise or flexible, particularly when a problem has a natural return associated with it that only needs adjustment in order to clarify what we wish to predict or optimize.

We then review previous work touching on estimating functions of the return, along with some proposed extensions. We discuss methods for estimating the moments of the return, and describe how to use these moments to approximate more general functions using parametric methods or Taylor series. We also provide an overview of distributional reinforcement learning, which provides an alternative way of approximating functions of the return.

3.1 The Reward Hypothesis in Practice

The *reward hypothesis* in reinforcement learning states that everything that we might want to optimize can be formulated as a return:

> *That all of what we mean by goals and purposes can be thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).*[1]

Assuming that achieving a goal is quantifiable[2], then the reward hypothesis is, if not self-evidently true, at least not immediately falsifiable.

[1] *Reinforcement Learning: An Introduction*, page 53. According to the chapter notes, it was first suggested by Michael Littman in a personal communication with the authors.

[2] That is, we have some way of representing success as a number– it might be binary (whether the goal was accomplished or not) or real-valued when there are degrees of success.

Sometimes the form that the rewards (and thus the return) should take is obvious. For example, in shortest path problems, the objective is to select a route that goes from one location to another with minimum total cost. The costs of each move map to the (negative) transition reward, and the return is then the undiscounted sum of these rewards. These sorts of problems are easier for agents to learn, because every transition is informative.

Other tasks are somewhat more difficult, because even if we know what the return should be, the rewards are sparse, *i.e.*, mostly zero, and so the agent receives no feedback over long periods of time. When it does receive a non-zero reward, it is hard to gauge which transitions were responsible for it. For example, in board games such as Chess or Go, the return might be one of $\{1, -1, 0\}$ depending if the agent wins, loses, or draws, provided by a reward on the final move. The reward for all the preceding moves is zero, because the outcome of the game is indeterminate until the end.

Board games can last for hundreds of moves, but they can be simulated efficiently, so sparsity issues can be overcome by training the agent through self-play at superhuman speeds. More complicated games, such as *DotA 2*, can last for tens of thousands of frames[3] before the outcome is reached; they are also less amenable to simulation. To make learning more tractable, an experimenter might modify the reward function to incentivize behavior they expect to be conducive to the underlying goal. Early efforts in computer Chess, for example, provided bonuses for capturing an opponent's pieces or placing the enemy king in check. These bonuses can conflict with the directive to win—material advantage is irrelevant if you ultimately lose the game—but sometimes a flawed reward function is better than one that is excessively sparse.

More complex or abstract goals may not even have an obvious return, let alone a clearly defined reward. For example:

- Producing a great work of art,

[3]The average game takes around thirty or forty minutes, with a 30Hz update rate, giving $54,000-72,000$ time-steps for a typical match.

- Investing funds to achieve a comfortable level of wealth,
- Playing a game in an entertaining fashion,
- Preparing a delicious meal,
- Generating an appropriate response to a question,
- Controlling a robot in a graceful manner.

Note the many adjectives employed in the tasks described above. We can formulate a return for a game-playing agent and teach it to win, but those victories might be cheapened if they are the result of a boring strategy that only a machine would have the patience to execute. The real purpose of playing a game is to have fun, but it is not clear how to express this[4] without recourse to external evaluation.

That is, we can still define a return that captures whether the agent has succeeded at the actual goal if we incorporate human consultation[5], although this might introduce problems of its own. Feedback can be subjective and inconsistent; even if an agreeable metric of success can be determined, collecting samples will be expensive[6]. These issues are not insurmountable, but can make training the agent too difficult or costly in practice.

So while the reward hypothesis remains intact, *our* goal to produce a successful learning agent may be infeasible depending on the problem in question.

As when dealing with sparse rewards, the usual approach is to modify the task specification to make it easier to learn, ideally without deviating too much from the original version of the problem. Typically, this involves coming up with a proxy for the underlying objective that is easier to measure or compute. This strategy is required so frequently that it is easy to conflate the proxy task with the original goal. However, differences between the real objective and the proxy can lead to unexpected or undesirable behavior from the learning agent.

For example, we might approximate the goal "generate an appropriate reply to a question" by having our agent answer queries on a social media website and reward

[4] *i.e.*, to put the"fun" in "function".

[5] To check whether our agent has produced a great work of art, we merely have to let it create something, and then wait decades or centuries in order to find out whether it was well-received.

[6] In terms of financial cost or the time and effort needed to acquire them.

it according to the feedback from that site's other users. Insofar as our definition of "appropriate" is based on approbation, this proxy is reasonable. However, if the real goal of our project was to correct misinformation, then this proxy might teach the agent to produce responses that merely *seem* correct, thereby exacerbating the problem we were trying to solve.

The problem of codifying our intentions such that learning is feasible while avoiding undesired behavior is of substantial practical and theoretical interest.

While we do not have a solution in general, we note that many interesting tasks have a natural reward associated with them, one that *almost* captures the spirit of what we wish to predict or optimize. We believe that in such cases, rather than using reward shaping or modifying the problem specification, it would be better to instead learn the obvious prediction target and then adjust it as circumstance demands. The strategy we have in mind is learning to estimate *functions of the return* of the form:

$$\psi_\pi(s) = \mathop{\mathbb{E}}_{\pi}[f(G_t)|S_t = s]. \tag{3.1}$$

This approach can avoid pitfalls arising from changing the task specification while allowing us to formulate our predictions and goals more precisely. It is distinct from merely modifying the value function after learning, as generally $\mathbb{E}_\pi\left[f(G_t)\right] \neq f(\mathbb{E}_\pi[G_t])$.

We provide some examples to illustrate how this could be useful:

Example: Route Selection If we were training an autonomous car to transport people around, then the obvious choice for the approximation target would be "time required to reach a destination"[7], corresponding to a reward of -1 for each time-step where the agent has not yet arrived at the goal. An agent trained with these rewards will learn to choose the fastest route, and its passengers will be happy insofar as they value their time. This setup is also conducive to learning: as the agent goes from the start to its destination, it can update its estimate for the travel time at

[7]Ignoring things like fuel costs, distance travelled, or (as seems fashionable) passenger safety.

each point along the way.

However, when travelling to an appointment where punctuality is crucial, the route that is fastest on average is not necessarily ideal, if it is prone to the occasional unpredictable delay. The risk of such a delay causing you to arrive late could outweigh the benefit of a few minutes saved. Rather, the better policy is one that ensures you will make it to your destination on time, even if it is a bit slower than the alternative.

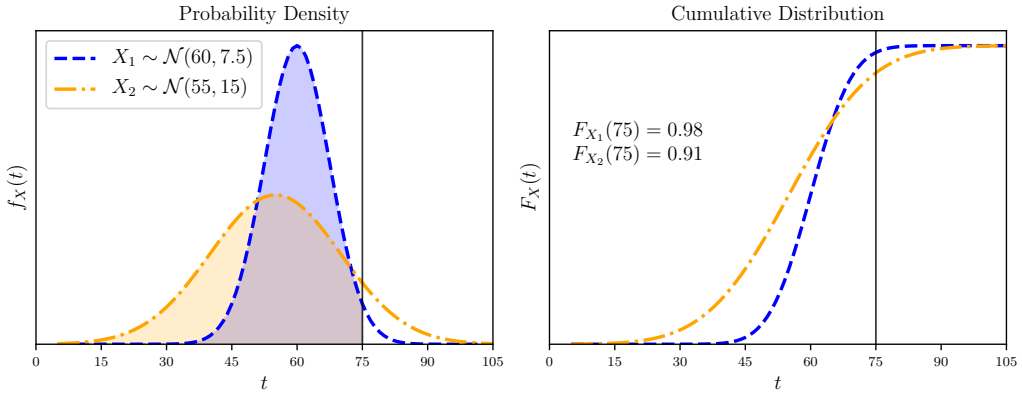

Figure 3.1: *An example of when optimizing for the best travel time on average can be counterproductive if there is an additional requirement to arrive before a certain time. Here we model two different routes as having normally distributed arrival times, with* $X_1 \sim \mathcal{N}(60, 7.5)$ *and* $X_2 \sim \mathcal{N}(55, 15)$*. The second route (corresponding to* X_2*) is quicker in expectation, but if we want to arrive in 75 minutes or less, we find that we will be late approximately 9% of the time. The first route is slower on average, but we will only be late about 2% of the time if we choose it instead.*

We could, of course, change the formulation of the problem to match the goal of "reaching the destination on time", perhaps by setting the reward to 1 if we arrive on schedule and 0 if we are late, or something along similar lines. To accommodate anxious travellers, the agent could perhaps keep track of how to make it from one place to another under a panoply of different time budgets using off-policy learning.

While we have salvaged the reward hypothesis, in practice this makes the problem significantly more complicated. The agent would have to learn policies for each possible time budget, possibly more depending on whether the travellers need to reach their destination within a certain time frame or would merely prefer to not be

unfashionably late.

If, instead, we kept the "natural" rewards, but learned to estimate the distribution of the return (rather than just its mean), we would be able to compute $\mathbb{E}_\pi[f(G_t)]$ for different choices of $f(\cdot)$. We believe that learning general facts about the environment and then adapting them to specific needs provides a more extensible, and potentially more efficient[8] approach to designing and training learning agents.

Example: Investing The goal of a trading algorithm might be to maximize its profit, and so an obvious choice for the reward would be the gain or loss from its trades. This simple reward function provides a reasonable baseline for measuring performance, but most market participants also care about the risk associated with their investments. Volatile securities might be a source of short-term profit, but over longer time-spans, assets with more consistent returns can produce higher growth.

One might object that this can be addressed by refining how we measure performance to take compounding into effect without having to use functions of the return. Even so, the existence of many different financial products which appeal to different investors points to a deeper issue: the point of investing is not so much maximizing the value of one's accounts so much as maximizing the utility derived from those funds[9]. Rather than learning a slightly different value function for each investor, we could instead learn to predict the possible outcomes for different asset allocations and then evaluate how desirable each choice is on an individual basis.

3.2 Why Not Just Directly Estimate?

If learning to predict or optimize functions of the return could be useful, why is this approach not in common use? Mainly, it is because this is difficult to do without removing the reasons we employ reinforcement learning in the first place.

[8] Insofar as estimating the distribution of the return can be done reliably and that a function of the distribution can be computed in a convenient manner.

[9] Quoting William Micawber in Dickens' *Great Expectations*, "Annual income twenty pounds, annual expenditure nineteen nineteen and six, result happiness. Annual income twenty pounds, annual expenditure twenty pounds ought and six, result misery." A budget shortfall can be extremely stressful, much more so than having an equivalent surplus is pleasant.

One key advantage offered by RL methods is that they can learn in the present, even when the ultimate outcome has yet to be revealed. That is, they can adapt by trial-and-error without requiring full knowledge of their actions' future effects.

The clearest way to see this is by following the math. First, note the return's recursive form:

$$\begin{aligned} G_t &\stackrel{\text{def}}{=} \sum_{n=0}^{\infty} R_{t+n+1} \prod_{k=1}^{n-1} \gamma_{t+k} = R_{t+1} + \gamma_{t+1} R_{t+2} + \gamma_{t+1}\gamma_{t+2} R_{t+3} + \cdots \\ &= R_{t+1} + \gamma_{t+1} G_{t+1}. \end{aligned} \tag{2.2}$$

If the environment is Markovian, then the value (expected return) of a given state under policy π can be expressed as a Bellman equation:

$$\begin{aligned} v_\pi(s) &\stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{\pi}[G_t | S_t = s] = \mathop{\mathbb{E}}_{\pi,s}[R_{t+1} + \gamma_{t+1} G_{t+1}] \\ &= \mathop{\mathbb{E}}_{\pi,s}[R_{t+1}] + \mathop{\mathbb{E}}_{\pi,s}[\gamma_{t+1} v_\pi(S_{t+1})]. \end{aligned} \tag{3.2}$$

Without knowing the true value function v_π, we can still estimate the value of a state using an approximation of the value function, say $\hat{v}$:

$$\begin{aligned} \hat{v}(s) &= \mathop{\mathbb{E}}_{\pi,s}[R_{t+1} + \gamma_{t+1} \hat{v}(S_{t+1})] \\ &\approx v_\pi(s). \end{aligned} \tag{3.3}$$

We could then modify $\hat{v}$ to make it more consistent with itself[10] thereby bringing it closer to v_π, or adjust π to improve the expected return, all without waiting to observe the rest of the reward sequence.

However, if we consider the expectation of a nonlinear function of the return, things become dicier. It is generally no longer possible to separate the expected return into a term for the current reward and a term associated with the return from the next state, as $f(G_t) \neq f(R_{t+1}) + f(\gamma_{t+1} G_{t+1})$ unless $f(\cdot)$ is linear.

Of course, we *could* just record the full return for each state, compute $f(G_t)$, and then approximate $u(s) = \mathbb{E}_\pi[f(G_t)|S_t = s]$ as if we were doing supervised learning. For the class of problems where we would like to use reinforcement learning[11], this

[10] For example, using temporal difference methods (see Section 2.5).

[11] Problems that benefit from online, incremental improvement, as well as those with computation

knowledge is cold comfort.

Unless we can somehow formulate $\mathbb{E}_\pi[f(G_t)]$ as a Bellman equation, it would seem we must either try a different sort of algorithm, or resign ourselves to learning something other than our preferred prediction target. Fortunately, we are not reduced to barbarity just yet.

3.3 Estimating Functions of the Return

Having argued for the desirability of learning to estimate functions of the return, we now proceed to describe how this might be done. Broadly speaking, there are two main lines of attack:

1. Learn the moments of the return, and use these to approximate functions of the return.
2. Learn to estimate the distribution of the return and evaluate the function on this distribution.

The first approach builds on the work of Sobel (1982), Tamar, Castro and Mannor (2016), White and White (2016), Sato, Kimura and Kobayashi (2002) and Munos and Moore (1999), while the distributional approach is a relatively recent development. We briefly describe both techniques in the following sub-sections.

3.3.1 Moment Estimation

The moments of a probability distribution provide a way to characterize that distribution in terms of how its probability mass is located relative to some point. The term "moment" itself derives from an analogy with physics[12], where the moment of inertia summarizes how the mass of a system is distributed. As in physics, the moments of a distribution depend on where you measure from; typically, we consider the moments either with reference to zero or to the distribution's mean.

or memory constraints.

[12] According to David (1995), it was introduced by Karl Pearson in a letter to *Nature* (Pearson 1893), wherein he describes a method of fitting curves to a binomial distribution.

The n-th moment (sometimes referred to as a *raw moment*) of a random variable X (denoted μ_n) is defined as

$$\mu_n \stackrel{\text{def}}{=} \mathbb{E}[X^n] = \int_{-\infty}^{\infty} x^n f_X(x) \mathrm{d}x, \tag{3.4}$$

where $f_X(\cdot)$ is the probability density function for X.

The first moment is just the expected value of X, and is common enough that we typically write μ in place of μ_1. These raw moments are somewhat difficult to interpret for $n \geq 2$, so it is common to instead consider the "higher moments" centered around the mean[13].

The n-th *central moment*, which we denote by c_n, is given by:

$$c_n \stackrel{\text{def}}{=} \mathbb{E}[(X - \mu)^n] = \int_{-\infty}^{\infty} (x - \mu)^n f_X(x) \mathrm{d}x \tag{3.5}$$

For $n = 2$, we have $c_2 = \mathrm{Var}[X]$, the variance of X. It can be thought of as a measure of dispersion for X. When the distribution is tightly concentrated around the mean, the variance is low; if most of the probability mass is far from the mean, then the variance will be higher.

The higher central moments measure the lopsidedness (for n odd) or tailedness of the distribution (when n is even). These can be useful to know, but for even knowing a few moments allows us to express a number of common distributions[14].

Although the central moments are a bit easier to interpret, sometimes it is easier to work with raw moments instead. The raw moments and the central moments are related by a binomial transformation, so if we know n moments of either type, we can convert between the two kinds:

$$\begin{aligned} c_n &= \sum_{k=0}^{n} \binom{n}{k} (-1)^{n-k} \mu_k \mu_1^{n-k}, \\ \mu_n &= \sum_{k=0}^{n} \binom{n}{k} c_k \mu_1^{n-k}. \end{aligned} \tag{3.6}$$

[13] In physics, for $n = 2$, the corresponding raw moment would be the moment of inertia with respect to the origin, which might be useful as part of some larger calculation. This is informative in probability theory, since the origin can be somewhat arbitrary; we define the raw moments with respect to zero primarily because it's a unique reference point on the real line.

[14] For example, we only need to know the mean and the variance to determine the form of a Gaussian distribution.

3.3.2 The Return's Second Moment as a Bellman Equation

The return is a random variable, so its second moment for state s can be expressed using (3.4) as:

$$v_\pi^{(2)}(s) \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{\pi}[G_t^2 | S_t = s]. \tag{3.7}$$

As observed by Sobel (1982), it is possible to formulate a Bellman equation for the second moment, provided we have access to the true value function v_π. To see this, we expand G_t^2 from its definition (2.2), yielding:

$$\begin{aligned} G_t^2 &= (R_{t+1} + \gamma_{t+1} G_{t+1})^2 \\ &= R_{t+1}^2 + 2\gamma_{t+1} R_{t+1} G_{t+1} + \gamma_{t+1}^2 G_{t+1}^2. \end{aligned} \tag{3.8}$$

Then, taking the expectation, we get:

$$\begin{aligned} v_\pi^{(2)}(s) = \mathop{\mathbb{E}}_{\pi,s}[G_t^2] &= \mathop{\mathbb{E}}_{\pi,s}[R_{t+1}^2 + 2\gamma_{t+1} R_{t+1} G_{t+1} + \gamma_{t+1}^2 G_{t+1}^2] \\ &= \mathop{\mathbb{E}}_{\pi,s}[R_{t+1}^2 + 2\gamma_{t+1} R_{t+1} G_{t+1}] + \mathop{\mathbb{E}}_{\pi,s}[\gamma_{t+1}^2 G_{t+1}^2] \\ &= \mathop{\mathbb{E}}_{\pi,s}[R_{t+1}^{(2)}] + \mathop{\mathbb{E}}_{\pi,s}[\gamma_{t+1}^2 v_\pi^{(2)}(S_{t+1})], \end{aligned} \tag{3.9}$$

where we use $R_{t+1}^{(2)} = R_{t+1}^2 + 2\gamma_{t+1} R_{t+1} G_{t+1}$. Observe that $R_{t+1}^{(2)}$ can be expressed as depending only the transition (s, s') using $v_\pi(\cdot)$. This is obviously true for the R_{t+1}^2 component, and for the cross-term we have:

$$\begin{aligned} \mathop{\mathbb{E}}_{\pi,s}[\gamma_{t+1} R_{t+1} G_{t+1}] &= \sum_{s'} p_\pi(s, s') \mathop{\mathbb{E}}_{\pi,s,s'}[\gamma_{t+1} R_{t+1} G_{t+1}] \\ &= \sum_{s'} p_\pi(s, s') \gamma(s') r(s, s') \mathop{\mathbb{E}}_{\pi,s,s'}[G_{t+1}] \\ &= \sum_{s'} p_\pi(s, s') \gamma(s') r(s, s') v_\pi(s'), \end{aligned} \tag{3.10}$$

where we first use the law of total expectation to expand $\gamma_{t+1} R_{t+1} G_{t+1}$, and note that, given the Markov property, the discount and rewards are independent of the subsequent return conditioned on s', allowing us to extract them from the expectation, so we can substitute $v_\pi(s')$ for $\mathbb{E}_{\pi,s,s'}[G_{t+1}]$.

If we define $r_\pi^{(2)} : \mathcal{S} \to \mathbb{R}$ via:

$$r_\pi^{(2)}(s) \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{\pi}[R_{t+1}^{(2)} | S_t = s] = \sum_{s'} p_\pi(s, s') \big[r^2(s, s') + 2\gamma(s') r(s, s') v_\pi(s') \big], \tag{3.11}$$

then we can write a Bellman equation for the second moment of the return:

$$v_\pi^{(2)}(s) = \mathop{\mathbb{E}}_{\pi}[G_t^2 | S_t = s] = r_\pi^{(2)}(s) + \sum_{s'} p_\pi(s, s') \gamma^2(s') v_\pi^{(2)}(s'). \tag{3.12}$$

It is then easy to express the variance of the return:

$$u_\pi(s) = \mathop{\mathbb{E}}_{\pi, s}[G_t^2] - \mathop{\mathbb{E}}_{\pi, s}[G_t] = v_\pi^{(2)}(s) - v_\pi^2(s). \tag{3.13}$$

Sobel also notes that u_π itself can be expressed as a Bellman equation:

$$\begin{aligned} m_\pi(s) &\stackrel{\text{def}}{=} -v_\pi^2(s) + \sum_{s'} P_\pi(s, s') (r(s, s') + \gamma(s') v_\pi(s'))^2, \\ u_\pi(s) &= m_\pi(s) + \sum_{s'} p_\pi(s, s') \gamma^2(s') u_\pi(s'). \end{aligned} \tag{3.14}$$

Later, in Section 4.3, we will show that (3.13) and (3.14) amount to the same thing.

The preceding equations assume that the true value function is available, which is not usually true in situations where reinforcement learning is called for. Developing a practical learning algorithm required some additional work, which we summarize in the next section.

3.3.3 Learning the Second Moment of the Return

Building on the work of Sobel (1982), Tamar, Castro and Mannor (2016) and White and White (2016) developed algorithms for estimating the second moment of the return and therefore the return's variance. Their approaches posit that, rather than using v_π in (3.11), we might instead use an *approximate* value function instead, which could learned in parallel.

For some arbitrary value function $v : \mathcal{S} \to \mathbb{R}$ and discount $\bar{\gamma} : \mathcal{S} \to [0, 1]$, we define:

$$\hat{R}_{t+1}^{(2)} \stackrel{\text{def}}{=} R_{t+1}^2 + 2\gamma_{t+1} R_{t+1} v(S_{t+1}), \tag{3.15}$$

$$\hat{G}_t^{(2)} = \hat{R}_{t+1}^{(2)} + \bar{\gamma}_{t+1} \hat{G}_{t+1}^{(2)} = \sum_{n=1}^{\infty} \hat{R}_{t+n}^{(2)} \prod_{k=1}^{n-1} \bar{\gamma}_{t+k}. \tag{3.16}$$

Note that $\widehat{G}_t^{(2)}$ is a valid return, and could be targeted by *any* reinforcement learning algorithm by using $R^{(2)}$ instead of R as the cumulant and $\bar{\gamma}$ in place of γ as the discount factor. If $v = v_\pi$ and $\bar{\gamma} = \gamma^2$, then $\mathbb{E}_\pi[\widehat{G}_t^{(2)}] = \mathbb{E}_\pi[G_t^2]$, so the algorithm would learn to approximate the return's second moment.

Then it would seem that learning to estimate the second moment of the return is just a matter of learning $v \approx v_\pi$ and making the appropriate substitutions to generate this new target. However, there is a potential hazard to this approach—note the tacit assumption in (3.16) that $v(\cdot)$ does not change. So if we aim to learn both v and $v^{(2)}$ *simultaneously*, $\mathbb{E}_\pi[\widehat{G}_t^{(2)}|S_t = s]$ will change over time, and we have to wonder whether this will cause our estimator to diverge.

As it turns out, using v in place of v_π can yield convergent approximation schemes even when v is non-constant. Tamar, Castro and Mannor (2016) show this by expressing the problem in terms of jointly estimating v and $v^{(2)}$ and demonstrating that the resulting system is convergent. They show that convergence holds even under linear function approximation; with an elegant argument based on norm equivalence they even avoid the need to employ separate time-scales for the estimators. Their results immediately lend themselves to variants of standard RL algorithms, such as the versions of TD(0) and LSTD they describe in their paper.

White and White (2016) extend the framework to learning the second moment of the λ-return, with the aim of adaptively setting λ to optimize the bias-variance trade-off for the value estimate[15]. The expressions they derive generalize earlier results significantly, accounting for variable γ and λ with straightforward extensions to off-policy estimation. They first expand from the definition of the λ-return:

$$\begin{aligned}(G_t^\lambda)^2 &= \left(R_{t+1} + \gamma_{t+1}[(1-\lambda_{t+1})v(S_{t+1}) + \lambda_{t+1}G_{t+1}^\lambda]\right)^2 \\ &= \underbrace{(R_{t+1}^\lambda)^2 + 2R_{t+1}^\lambda\gamma_{t+1}\lambda_{t+1}G_{t+1}^\lambda}_{\bar{R}_{t+1}^{(2)}} + \gamma_{t+1}^2\lambda_{t+1}^2(G_{t+1}^\lambda)^2,\end{aligned} \tag{3.17}$$

where $R_{t+1}^\lambda \stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}(1-\lambda_{t+1})v(S_{t+1})$. Taking the expectation as in (3.9)

[15]Recall that higher λ tends to reduce the bias for the TD(λ) fixed-point, but at the cost of increasing the variance of the approximation target; its overall error is the combination of the bias squared and the variance.

and (3.10), we get:

$$\begin{aligned}
\underset{\pi}{\mathbb{E}}[\bar{R}^{(2)}_{t+1}] &= \underset{\pi}{\mathbb{E}}[(R^{\lambda}_{t+1})^2 + 2R^{\lambda}_{t+1}\gamma_{t+1}\lambda_{t+1}G^{\lambda}_{t+1}] \\
&= \underset{\pi}{\mathbb{E}}[R^2_{t+1} + 2\gamma_{t+1}(1-\lambda_{t+1})R_{t+1}v(S_{t+1}) + \gamma^2_{t+1}(1-\lambda_{t+1})^2v^2(S_{t+1})] \quad (3.18) \\
&\quad + 2\,\underset{\pi}{\mathbb{E}}[\gamma_{t+1}\lambda_{t+1}R_{t+1}v_{\lambda}(S_{t+1}) + \gamma^2_{t+1}\lambda_{t+1}(1-\lambda)v_{\lambda}(S_{t+1})v(S_{t+1})],
\end{aligned}$$

where $v_{\lambda}(s) \overset{\text{def}}{=} \mathbb{E}_{\pi,s}[G^{\lambda}_t]$ is the expected value of the λ-return using value function v. If we assume that $v_{\lambda}(s) \approx v(s)$ and make the appropriate substitutions, we get:

$$\begin{aligned}
\underset{\pi}{\mathbb{E}}[\bar{R}^{(2)}_{t+1}] &\approx \underset{\pi}{\mathbb{E}}[R^2_{t+1} + 2\gamma_{t+1}(1-\lambda_{t+1})R_{t+1}v(S_{t+1}) + \gamma^2_{t+1}(1-\lambda_{t+1})^2v^2(S_{t+1})] \\
&\quad + 2\,\underset{\pi}{\mathbb{E}}[\gamma_{t+1}\lambda_{t+1}R_{t+1}v(S_{t+1})\gamma^2_{t+1}\lambda_{t+1}(1-\lambda)v^2(S_{t+1})] \quad (3.19) \\
&= \underset{\pi}{\mathbb{E}}[R^2_{t+1} + 2\gamma_{t+1}R_{t+1}v(S_{t+1}) + \gamma^2_{t+1}(1-\lambda^2_{t+1})v^2(S_{t+1})],
\end{aligned}$$

which allows us to define versions of (3.15) and (3.16) for the λ-return:

$$\hat{R}^{(2),\lambda}_{t+1} = R^2_{t+1} + 2\gamma_{t+1}R_{t+1}v(S_{t+1}) + \gamma^2_{t+1}(1-\lambda^2_{t+1})v^2(S_{t+1}) \quad (3.20)$$

$$\hat{G}^{(2),\lambda}_t = \hat{R}^{(2),\lambda}_{t+1} + \bar{\gamma}_{t+1}\hat{G}^{(2),\lambda}_{t+1} = \sum_{n=1}^{\infty}\hat{R}^{(2),\lambda}_{t+n}\prod_{k=1}^{n-1}\bar{\gamma}_{t+k}. \quad (3.21)$$

For (3.21) we get $\mathbb{E}_{\pi}[\hat{G}^{(2),\lambda}_t] = \mathbb{E}_{\pi}[(G^{\lambda}_t)^2]$ if $v = v_{\lambda}$ and $\bar{\gamma} = \gamma^2\lambda^2$. In particular, with $\lambda = 1$ and $\bar{\gamma} = \gamma^2$, we recover (3.15) and (3.16), which is to say that $\hat{G}^{(2),\lambda=1}_t$ is an approximation for the Monte Carlo return's second moment.

Note that both $\hat{R}^{(2),\lambda}_{t+1}$ and $\hat{G}^{(2),\lambda}_{t+1}$ are still well-defined even if $v_{\lambda} \neq v$, or if $\bar{\gamma} \neq \gamma^2\lambda^2$, although we would no longer be estimating the second moment of the λ-return. In practice, the value function learned by an RL agent will approximate v_{λ}, so substituting v for v_{λ} is not unreasonable. Furthermore, since v_{λ} is usually just a proxy for v_{π}[16], then we could argue that any reasonably accurate value function could be used when defining (3.20).

As White and White (2016) and Sherstan et al. (2018) point out, this gives us the ability to estimate the second moment of the λ-return using a value function learned with a *different* amount of bootstrapping[17], so long as v is "close enough" to

[16] Absent function approximation, the fixed-point for v_{λ} *is* v_{π}.

[17] Or even to estimate the second moment of *multiple* λ-returns using a *single* value function.

v_λ. Tamar, Castro and Mannor (2016) do this implicitly, describing an algorithm which learns the value function via TD(0), but targets the second moment of the Monte Carlo return.

We provide one possible version of the algorithm (which we refer to as VTD) in Appendix C, listed as Algorithm 3. That algorithm has three bootstrapping hyperparameters: one for the base component, which uses TD(λ) to learn the value function; another for the second moment estimator, which is also TD(λ) based but with a different cumulant); and the third hyperparameter, which is used to *define* the cumulant as in (3.20).

We close this subsection by noting an alternative approach for estimating the variance in terms of the temporal difference errors, as observed by Munos and Moore (1999) and formulated as TD-style algorithm by Sato, Kimura and Kobayashi (2002). Using the squared TD error as a cumulant can yield a target for the return's variance, as we will show in Chapter 4. Tamar, Castro and Mannor (2016) note that the issue with this approach is that it leads to a non-linear equation for the variance, which makes analyzing the algorithm substantially harder. In later chapters we will show how some of these difficulties can be overcome; we provide expressions for the approximation target's fixed-point and prove that the algorithm converges.

3.3.4 Learning Higher Moments of the Return

Building on Section 3.3.3, we note that we can in fact construct approximation targets for higher moments of the return. The formula for the n-th power of the return is:

$$\begin{aligned} G_t^n &\stackrel{\text{def}}{=} (R_{t+1} + \gamma_{t+1} G_{t+1})^n \\ &= R_{t+1}^{(n)} + \gamma_{t+1}^n G_{t+1}^n, \end{aligned} \tag{3.22}$$

where

$$R_{t+1}^{(n)} \stackrel{\text{def}}{=} \sum_{k=0}^{n-1} \binom{n}{k} \gamma_{t+1}^k R_{t+1}^{n-k} G_{t+1}^k. \tag{3.23}$$

So the n-th moment of the return can be written as $\mathbb{E}[R_{t+1}^{(n)}] + \gamma_{t+1}^n \mathbb{E}[G_{t+1}^n]$, which is similar to a Bellman equation, although $R_{t+1}^{(n)}$ includes terms that contain powers of G_{t+1}, so it does not quite have the desired recursive form.

However, as in Section 3.3.3, we can use estimates of $G_{t+1}^{(n)}$ to produce an approximation target suitable for online estimation. If we have an estimate for the value function of the first $n-1$ moments (denoted $v^{(n)}(\cdot)$, with $v^{(n)}(s) \approx \mathbb{E}_\pi[G_t^n | S_t = s]$, we can define a new target for the *next*, $\hat{G}^n$:

$$\begin{aligned} \hat{G}^n &\stackrel{\text{def}}{=} \left(\sum_{k=0}^{n-1} \binom{n}{k} \gamma^k R_{t+1}^{n-k} v^{(k)}(S_{t+1}) \right) + \gamma^n \hat{G}_{t+1}^n \\ &= \hat{R}_{t+1}^{(n)} + \gamma^n \hat{G}_{t+1}^n. \end{aligned} \tag{3.24}$$

Here, $\hat{R}_{t+1}^{(n)}$ is given by:

$$\hat{R}_{t+1}^{(n)} \stackrel{\text{def}}{=} \sum_{k=0}^{n-1} \binom{n}{k} \gamma^k R_{t+1}^{n-k} v^{(k)}(S_{t+1}). \tag{3.25}$$

This new return is similar to (3.22), except we use $v^{(n)}(S_{t+1})$ in place of G_{t+1}^n in the binomial sum.

Ideally, we would have $\mathbb{E}_\pi[\hat{G}_t^n] = \mathbb{E}_\pi[G_t^n]$. If our estimated value functions $v^{(k)}(s) = \mathbb{E}_\pi[G_t^k | S_t = s]$ for $k = 1, 2, \ldots, n-1$, then this is indeed the case. However, because each target is defined in terms of the previous approximate value functions, inaccuracies can be magnified and learning more than the first few moments using this approach can be difficult.

Nonetheless, even if our value functions for the lower moments are only approximate, then $\hat{G}_t^n$ is still well-defined, but will probably be a biased estimate of the true G_t^n. Following the example set by Tamar, Castro and Mannor (2016) and White and White (2016), we note that a stable algorithm might still be derived. For the curious, we provide pseudocode for such an algorithm (dubbed MOMENT-TD), given in Algorithm 4 (Page 116) and originally reported by Bennett et al. (2019). It allows for estimating arbitrary moments of the return, although at this point it has not been studied in detail[18].

[18]Partly this might be because of a lack of immediate usefulness, as trying to learn the (non-central)

3.3.5 Taylor Series of a Random Variable

The *Taylor series* is an expansion of a function around a point. For example, for a function $g : \mathbb{R} \to \mathbb{R}$, the Taylor series about $a \in \mathbb{R}$ is

$$\begin{aligned} f(x) &= f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^{2!} + \frac{f^{(3)}}{3!}(x-a)^3 + \cdots \\ &= \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k, \end{aligned} \tag{3.26}$$

where $f^{(k)}(a)$ denotes $\frac{\mathrm{d}^k f}{\mathrm{d}x^k}$ evaluated at a.

If $g : \mathbb{R} \to \mathbb{R}$ is a function whose Taylor series converges to $g(x)$ when evaluated at any $a \in \mathbb{R}$, then we can express the expected value of a function of a random variable using a Taylor series:

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x) \mathrm{d}x = \int_{-\infty}^{\infty} \left[\sum_{k=0}^{\infty} \left(\frac{g^{(k)}(a)}{k!} \right) (x-a)^k \right] f_X(x) \mathrm{d}x, \tag{3.27}$$

where $f_X(\cdot)$ is the probability density function for X.

We can get a more useful form if we can switch the order of summation and integration. Let $h_k(x) = \left(\frac{g^{(k)}(a)}{k!} \right) (x-a)^k f_X(x)$. If $\int_{-\infty}^{\infty} \sum_{k=0}^{\infty} |h_k(x)| < \infty$, the the Fubini-Tonelli theorem applies, and we can swap the order of the operations:

$$\int_{-\infty}^{\infty} \sum_{k=0}^{\infty} h_k(x) \mathrm{d}x = \sum_{k=0}^{\infty} \int_{-\infty}^{\infty} h_k(x) \mathrm{d}x, \tag{3.28}$$

that is,

$$\sum_{k=0}^{\infty} \int_{-\infty}^{\infty} \left(\frac{g^{(k)}(a)}{k!} \right) (x-a)^k f_X(x) \mathrm{d}x = \sum_{k=0}^{\infty} \left(\frac{g^{(k)}(a)}{k!} \right) \int_{-\infty}^{\infty} (x-a)^k f_X(x) \mathrm{d}x, \tag{3.29}$$

where we can extract $\frac{g^{(k)}(a)}{k!}$ from the integral because it is constant given a.

For our purposes, there are two natural choices for a: $a = 0$, or $a = \mu_1$. If $a = 0$,

moments of the return tends to be stymied by numerical error. Our preliminary investigations indicate that MOMENT-TD is reasonably accurate for the first few moments, but begins to degrade for $n > 4$ or so.

then:

$$\begin{aligned}\mathbb{E}[g(X)] &= \sum_{k=0}^{\infty}\left(\frac{g^{(k)}(0)}{k!}\right)\int_{-\infty}^{\infty} x^k f_X(x)\mathrm{d}x \\ &= \sum_{k=0}^{\infty}\left(\frac{g^{(k)}(0)}{k!}\right)\mu_k.\end{aligned} \tag{3.30}$$

If $a = \mu_1$, we get:

$$\begin{aligned}\mathbb{E}[g(X)] &= \sum_{k=0}^{\infty}\left(\frac{g^{(k)}(\mu_1)}{k!}\right)\int_{-\infty}^{\infty} (x-\mu_1)^k f_X(x)\mathrm{d}x \\ &= \sum_{k=0}^{\infty}\left(\frac{g^{(k)}(\mu_1)}{k!}\right)c_k.\end{aligned} \tag{3.31}$$

The utility of this approach is somewhat blunted by the fact that it requires $g(\cdot)$ to have an infinite radius of convergence, and the further requirement that X must have finite moments[19]. For stochastic approximation, it is more concerning that we do not typically have access to the true moments, meaning that even if X and $g(\cdot)$ are well-behaved, our estimate for $\mathbb{E}[g(X)]$ from the Taylor expansion may be wrong if our moment estimates are off.

Still, if we have a reasonable estimate for the mean and variance of the return, then this gives us a quick way to approximate the expected value of $g(G_t)$. Noting that $c_1 = \mathbb{E}[X - \mathbb{E}[X]] = 0$, we can use:

$$\begin{aligned}\mathbb{E}_{\pi,s}[g(G_t)] &\approx \sum_{k=0}^{\infty}\left(\frac{g^{(k)}(v(s))}{k!}\right)c_k(s) = g(v(s)) + \cancelto{0}{g'(v(s))c_1(s)} + \frac{1}{2}g''(v(s))u(s) + \cdots \\ &\approx g(v(s)) + \frac{1}{2}g''(v(s))u(s),\end{aligned} \tag{3.32}$$

which is straightforward to compute and obviates the need to express $g(G_t)$ as a return.

3.3.6 Parametric Approximation

If we have the first n moments of a random variable, we might approximate it by some parameterized distribution. We can then estimate the the expected value of a function of that random variable using the parametric approximation as a stand-in.

Suppose we wanted to estimate $\mathbb{E}[g(X)]$ for some function $g : \mathbb{R} \to \mathbb{R}$ and X a

[19] This is implicit in the requirement that $\int \sum |h_n(x)| \mathrm{d}x < \infty$.

random variable. The precise distribution for X might be unknown, but it might be similar enough to another distribution, allowing us to approximate X by a random variable, say $\hat{X}$, from the known distribution. Then:

$$\mathbb{E}[g(X)] \approx \mathbb{E}[g(\hat{X})] = \int_{-\infty}^{\infty} p_{\hat{X}}(x) g(x) \mathrm{d}x. \tag{3.33}$$

For example, given the mean and variance of X (μ and c_2, respectively), we might approximate X as normally distributed, with $X \approx \hat{X} \sim \mathcal{N}(\mu, c_2)$. We could then estimate

$$\mathbb{E}[g(X)] \approx \frac{1}{\sqrt{2\pi c_2}} \int_{-\infty}^{\infty} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{c_2}\right\} g(x) \mathrm{d}x,$$

which is relatively straightforward to calculate.

The validity of this approach is dependent on the distribution used for approximation, and in practice also on the accuracy of the values used in the parameterization. If we have reason to believe that X is distributed in a particular way, then it is possible to achieve a close approximation with relatively scant information. Conveniently, many distributions can be parameterized in terms of their moments (like the normal distribution used in the example above).

On the other hand, choosing a distribution for the approximation entails greater complexity. In particular, part of the allure of TD methods is that they are "model-free", so by assuming that the returns have a certain form we are implicitly imposing a model on the environment.

3.4 Distributional Reinforcement Learning

Distributional Reinforcement Learning is a relatively recent development in RL, which seeks to expand what can be learned and approximated beyond the expected return by considering the *distribution* of returns. In exchange for the increased complexity this entails, it has a number of advantages, such as allowing us to better characterize the possible results of a given policy, and it seems to interact beneficially with function approximation using deep neural networks.

Our interest in this topic comes mainly from the fact that estimating the vari-

ance of the return is straightforward if we have an approximation of the return's distribution. Although estimating variance is not their primary use, distributional RL algorithms exhibit state-of-the-art performance on various benchmarks, so we would be remiss if we did not consider them in our comparisons.

The distributional RL algorithm we focus on is the C51 algorithm[20] introduced by Bellemare, Dabney and Munos (2017). The paper had a substantial impact because it suggested that learning the distribution of the return could lead to better performance, particularly in control tasks, and provided evidence for that claim in the form of state-of-the-art results on the Arcade Learning Environment testbed.

We restrict ourselves to studying C51 instead of performing a more complete survey of distributional methods, in part because it was the original distributional algorithm[21], but also due to concerns about scope and applicability. More recent algorithms might perform better than C51 on various benchmarks, but some of them are not quite "distributional" in the sense of learning a proper probability distribution[22]. Since we are interested in estimating the variance of the return, we prefer to deal with C51, which has a straightforward probabilistic interpretation, and let it serve as an exemplar of distributional methods.

3.4.1 Mathematical Framework

We provide a summary of distributional RL, referring heavily to Lyle, Bellemare and Castro (2019) who provide a good overview of the formalism. The notation might be somewhat jarring for those mostly familiar with traditional reinforcement learning, but this excursion is brief and mostly self-contained so hopefully the effect is not excessively confusing.

[20]Because it is a *categorical* learning algorithm, and the original implementation by Bellemare, Dabney and Munos (2017) suggested a support of *51* atoms for the distribution approximation. Some recent papers also refer to it as CDRL, for "categorical distributional reinforcement learning". We will go along with the original nomenclature, because CDRL seems more apt to describe a class of methods, rather than the particular algorithm we investigate.

[21]As noted by Bellemare et al., other work has considered reinforcement learning in terms of distributions of the return, but usually for some secondary purpose, rather than as a task in itself.

[22]For example, the S51 algorithm introduced by Bellemare, Le Roux et al. (2019) is still referred to as a distributional algorithm, but the distribution it learns is not necessarily a *probability* distribution. S51 can assign negative weight to possible values of the return, and the weights do not necessarily sum to one, so we cannot interpret the weights as corresponding to classical probabilities.

We use "$\overset{\mathrm{D}}{=}$" to express distributional equality, mainly when drawing attention to the fact that the equation in question is in terms of distributions rather than other quantities. Denoting by $Z_\pi : \mathcal{S} \to \mathrm{Dist}(\mathbb{R})$ the *return distribution function* (see Bellemare, Dabney and Munos 2017) under policy π, defined as:

$$Z_\pi(s) \overset{\mathrm{D}}{=} \sum_{t=0}^{\infty} \gamma^t R(S_t) \qquad S_0 = s, S_{t+1} \sim P_\pi(\cdot|S_t), \tag{3.34}$$

where $R(s)$ is the distribution of rewards given state s.

The distributional version of the Bellman equation is:

$$Z_\pi(s) \overset{\mathrm{D}}{=} R(s) + \gamma Z_\pi(S'), \qquad S' \sim P_\pi(\cdot|s) \tag{3.35}$$

with S' a random variable corresponding to the subsequent state[23]. This in turn leads to the corresponding distributional Bellman operator:

$$\mathcal{T}_\pi Z_\pi(s) \overset{\mathrm{D}}{=} R(s) + \gamma P_\pi Z_\pi(S). \tag{3.36}$$

If we were working directly with the distributions, then the fact that the distributional Bellman operator (3.36) is a contraction (Bellemare, Dabney and Munos 2017; Rowland et al. 2018) implies that it has a fixed-point, which could be found by iteration. Sadly, this is not a situation we frequently encounter— typically, we are unable to represent arbitrary distributions exactly.

To make this an actionable problem, we have to have some way of approximating arbitrary probability distributions. One method of approximating a distribution (and the one used by C51) is as a *categorical distribution*, which is to say as a finite set of points which correspond to the possible values the approximation can represent. To make this precise we first have to describe some additional concepts.

The *Dirac delta distribution*, sometimes referred to as the Dirac delta function[24], can be formulated as the derivative of the Heaviside step function:

$$\delta(x) \overset{\text{def}}{=} \frac{\mathrm{d}}{\mathrm{d}x} H(x). \tag{3.37}$$

[23] We omit state-dependent discounting primarily because including it is superfluous in our limited analysis, and to define it properly would complicate the explanation to no real benefit.

[24] Although some would argue that this terminology is incorrect, as depending on one's definition it may not be a proper function.

There is some unfortunate overlap in notation between the Dirac delta and the temporal-difference error, but for the most part they are used in different contexts so we defer to convention by using δ for both.

It can be thought of as a distribution sharply peaked at a point,

$$\delta(x) = \begin{cases} +\infty & \text{if } x = 0 \\ 0 & \text{if } x \neq 0 \end{cases} \tag{3.38}$$

and normalized such that

$$\int_{-\infty}^{\infty} \delta(x) \mathrm{d}x = 1. \tag{3.39}$$

We can now define the categorical distributions we will use to approximate the distribution of the return. Such a distribution can be formalized as a weighted combination of (fixed) Dirac deltas. Let $z_1 < z_2 < \cdots z_K$ for some $K \geq 1$, with $z_i \in \mathbb{R}$ for $i = 1, 2, \ldots, K$. Then we denote the set of possible distributions with support $\mathbf{z}$ as $\mathcal{Z}$, defined by:

$$\mathcal{Z} = \left\{ \sum_{i=1}^{K} p_i \delta(x - z_i) \ : p_i \geq 0, \quad \sum_{i=1}^{K} p_i = 1 \right\}. \tag{3.40}$$

We might write the probability density function for such a distribution as:

$$f_Z(x) = \sum_{i=1}^{K} p_i \delta(x - z_i). \tag{3.41}$$

Computing the expected value is straightforward:

$$\mathbb{E}[Z] = \int_{-\infty}^{\infty} x f_Z(x) \mathrm{d}x = \int_{-\infty}^{\infty} x \sum_{i=1}^{K} p_i \delta(x - z_i) \mathrm{d}x = \sum_{i=1}^{K} z_i p_i, \tag{3.42}$$

where we use the fact that $\int_{-\infty}^{\infty} \delta(x - y) f(x) \mathrm{d}x = f(y)$. We get something similar for variance, letting $\bar{z} = \mathbb{E}[Z]$, we find

$$\mathbb{V}[Z] = \int_{-\infty}^{\infty} (x - \bar{z})^2 f_Z(x) \mathrm{d}x = \sum_{i=1}^{K} (z_i - \bar{z})^2 p_i. \tag{3.43}$$

In fact, we can compute the expected value of arbitrary functions of Z, although for our purposes the usefulness of this approach depends on how accurately it approximates Z_π.

3.4.2 The C51 Algorithm

The categorical representation is essentially equivalent to the tabular setting discussed previously— we are assuming that we can use a separate distribution for each state. Typically, we have to introduce some form of function approximation for this method to be practical. Although earlier work has mainly used deep neural nets for this purpose, it is entirely possible to run C51 using linear function approximation.

Let $\hat{Z} : \mathcal{S} \to \text{Dist}(\mathbb{R})$ denote the parameterized approximation for the return's distribution. Let $\mathbf{x}(\cdot)$ be a function $\mathbf{x} : \mathcal{S} \to \mathbb{R}^d$ that maps states to features, and let $\mathbf{W}^{K \times d}$ be C51's weight matrix. Then we use the softmax function to assign probabilities to the possible values of $\mathbf{z}$ based on $\mathbf{x}(s)$, via

$$\mathbf{p}(s) = \frac{e^{\mathbf{W}\mathbf{x}(s)}}{\left\| e^{\mathbf{W}\mathbf{x}(s)} \right\|_1}, \tag{3.44}$$

with the interpretation that $\hat{Z}(s)$ takes on value z_i with probability $p_i(s)$, the i-th entry of $\mathbf{p}(s)$.

The use of the softmax function guarantees that the probabilities are positive (due to the exponential), and that they sum to one (because of its inherent normalization). Alternative parameterizations that do not enforce these guarantees may still result in a functioning algorithm[25], but not one that can be used for estimating functions of the return, so they are less useful for estimating functions of the return.

Now that we have a means of representing arbitrary distributions in an approximate fashion, we turn to the question of learning the distribution of the return via reinforcement learning.

This is actually a fairly difficult problem. There are some complications from using a categorical approximation for the distribution, and we will also be using function approximation, further muddying the waters. It gets positively murky when we consider that rather than having direct access to the distribution of R and S' (as in (3.35)), we tend to only learn through *samples* of the form (s, a, r, s'). All this

[25] In the sense that they arrive at a good control policy or value function.

before we even consider what loss function to use or update rule to apply!

To explain how these issues were resolved would essentially recapitulate Bellemare, Dabney and Munos (2017), so instead we provide an outline of their approach.

1. Given a transition (s, a, r, s'), compute $\hat{Z}(s)$ and $\hat{Z}(s')$,
2. Create a pseudo-distribution $\hat{\mathcal{T}}\hat{Z} = r + \gamma\hat{Z}(s')$,
3. Project the probability masses from $\hat{\mathcal{T}}\hat{Z}$ onto the support $\mathbf{z}$ using the Cramér projection, yielding $\Pi_C\hat{\mathcal{T}}\hat{Z}$,
4. Update the parameters of $\hat{Z}$ using the gradient of the cross-entropy loss between $\Pi_C\hat{\mathcal{T}}\hat{Z}$ and $\hat{Z}(s)$.

The process of computing $\Pi_C\hat{\mathcal{T}}\hat{Z}$ is also shown in Fig. 3.2.

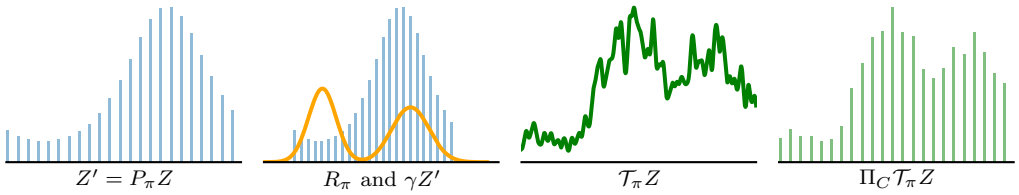

Figure 3.2: *The projected distributional Bellman operator can be computed by taking an estimate of the next state's return distribution ($P_\pi Z$), applying the discount γ to it and combining it with the reward to get $\mathcal{T}_\pi Z$, which is then projected onto the support $\mathbf{z}$ to produce $\Pi_C \mathcal{T} Z$.*

In contrast to other methods, there does not seem to be a closed-form expression for the fixed-point of C51[26]; however, it is interesting to note that in the LFA case its update equation resembles TD(0) applied to distributions. Pseudocode for the C51 algorithm under linear function approximation is given in Algorithm 5, and a derivation of the update rule is provided in Appendix B.

[26]However, we can generate a close approximation for its fixed-point through iterative methods.

Chapter 4

The δ^2-return and Variance

In this chapter we examine a method for learning the variance of the return directly, in contrast to the raw moment-based estimation used by VTD[1].

We show how the squared temporal difference errors can be used as a cumulant[2] with a well-defined return. If value function is exact (*i.e.*, $v = v_\pi$), then the δ^2-return corresponds to $\mathbb{E}_\pi[(G_t - v_\pi(S_t))^2]$, the variance of the return, and is in fact equivalent to the raw moment based approaches, albeit with more favorable numerical properties. We also discuss the more typical case where we do not have access to the true value function. In Section 4.5, we describe DVTD, an algorithm that targets the δ^2-return and provide an argument for its convergence..

4.1 Variance and the δ^2-return

In analogy with the value function, which maps states to their expected return, we might also imagine a variance function:

$$u_\pi(s) = \mathop{\mathbb{E}}_{\pi}\left[(G_t - v_\pi(s))^2 | S_t = s\right] = \underbrace{\mathop{\mathbb{E}}_{\pi}\left[G_t^2 | S_t = s\right]}_{v_\pi^{(2)}(s)} - \underbrace{\mathop{\mathbb{E}}_{\pi}\left[G_t | S_t = s\right]^2}_{v_\pi^2(s)}. \tag{4.1}$$

Earlier, in Section 3.3.3, we discussed how the second moment can be expressed as a Bellman equation, which in turn allows us to compute the variance as $u_\pi(s) =$

[1] See Section 3.3, particularly Section 3.3.3.
[2] *i.e.*, a reward-like signal.

$v_\pi^{(2)}(s) - v_\pi^2(s)$. However, the form of $G_t - v_\pi(s)$ suggests an alternative approach:

$$\begin{aligned}
u_\pi(s) &= \mathop{\mathbb{E}}_{\pi}[(G_t - \mathbb{E}[G_t])^2 | S_t = s] = \mathop{\mathbb{E}}_{\pi,s}[(G_t - v_\pi(s))^2] \\
&= \mathop{\mathbb{E}}_{\pi,s}[(R_{t+1} + \gamma_{t+1} G_{t+1} - v_\pi(s))^2] \\
&= \mathop{\mathbb{E}}_{\pi,s}\left[\left(R_{t+1} + \gamma_{t+1} v_\pi(S_{t+1}) - v_\pi(s) + \gamma_{t+1}(G_{t+1} - v_\pi(S_{t+1}))\right)^2\right] \\
&= \mathop{\mathbb{E}}_{\pi,s}\left[\left(\delta_t^\pi + \gamma_{t+1}(G_{t+1} - v_\pi(S_{t+1}))\right)^2\right] \\
&= \mathop{\mathbb{E}}_{\pi,s}[(\delta_t^\pi)^2 + 2\delta_t^\pi \gamma_{t+1}(G_{t+1} - v_\pi(S_{t+1})) + \gamma_{t+1}^2 (G_{t+1} - v_\pi(S_{t+1}))^2] \\
&= \mathop{\mathbb{E}}_{\pi,s}[(\delta_t^\pi)^2] + 2 \mathop{\mathbb{E}}_{\pi,s}[\delta_t^\pi \gamma_{t+1}(G_{t+1} - v_\pi(S_{t+1}))] + \mathop{\mathbb{E}}_{\pi,s}[\gamma_{t+1}^2 u_\pi(S_{t+1})],
\end{aligned} \tag{4.2}$$

where we use the superscript π in $\delta_t^\pi = R_{t+1} + \gamma_{t+1} v_\pi(S_{t+1}) - v_\pi(S_t)$ to draw attention to the fact that the value function used is the true value function, $v_\pi(\cdot)$.

As with the expansion for the return's second moment (see Section 3.3.2), the cross-term prevents us from immediately identifying (4.2) as a Bellman equation. But as it turns out, the cross-term $2\delta_t^\pi \gamma_{t+1}(G_{t+1} - v_\pi(S_{t+1}))$ is zero in expectation, thus:

$$u_\pi(s) = \mathop{\mathbb{E}}_{\pi}[\delta_t^2 + \gamma_{t+1}^2 u_\pi(S_{t+1}) | S_t = s]. \tag{4.3}$$

We can actually state a more general result that applies to the λ-return in addition to the Monte Carlo return:

Theorem 4.1 (Variance of the λ-return from TD Errors)

Let $G_t^{\lambda,\pi} = R_{t+1} + \gamma_{t+1}(\lambda_{t+1} G_{t+1}^{\lambda,\pi} + (1 - \lambda_{t+1}) v_\pi(S_{t+1}))$ be the λ-return given the value function $v(s) = v_\pi(s) = \mathbb{E}_\pi[G_t]$, and let $\delta_t^\pi = R_{t+1} + \gamma_{t+1} v_\pi(S_{t+1}) - v_\pi(S_t)$.

Then:

$$\begin{aligned}
u_\pi^\lambda(s) &\stackrel{\text{def}}{=} \mathop{\mathbb{V}}_{\pi}[G_t^{\lambda,\pi} | S_t = s] = \mathop{\mathbb{E}}_{\pi}\left[(\delta_t^\pi)^2 + \gamma_{t+1}^2 \lambda_{t+1}^2 (G_{t+1}^{\lambda,\pi} - v_\pi(S_{t+1}))^2\right] \\
&= \delta_\pi^2(s) + \sum_{s'} p_\pi(s, s') \gamma^2(s') \lambda^2(s') u_\pi^\lambda(s'),
\end{aligned} \tag{4.4}$$

where $\delta_\pi^2(s) = \mathbb{E}_\pi[(\delta_t^\pi)^2 | S_t = s]$.

Note that when $\lambda(s) = 1$ for all states, the λ-return is equivalent to the Monte

Carlo return, and (4.4) reduces to (4.3).

Before proving Theorem 4.1, we first justify our claim about the cross-term.

Lemma 4.2

Let $\{C_t\}_{t=0}$ and $\{B_t\}_{t=0}$ be sequences of random variables, with:

- $C_t = c(S_t, A_t, R_t, S_{t+1}) + \epsilon_t$, where $c : \mathcal{S} \times \mathcal{A} \times \mathcal{R} \times \mathcal{S} \to \mathbb{R}$ is a bounded function mapping transitions to real numbers, and $\{\epsilon_t\}$ is a noise sequence with mean zero,
- $B_t = G_t^\lambda - v_\pi(S_t)$.

Then:

$$\mathop{\mathbb{E}}_{\pi}[C_t B_{t+1} | S_t = s] = 0. \tag{4.5}$$

Proof (Lemma 4.2):

We prove this by simply expanding the expectation, noting that if we condition on the transition $(S_t, A_t, R_{t+1}, S_{t+1})$, then C_t and B_{t+1} are conditionally independent, with $\mathbb{E}_{\pi,s}[B_t] = \mathbb{E}_{\pi,s}[G_{t+1}^\lambda - v_\pi(S_{t+1})] = 0$.

First, observe that:

$$\mathop{\mathbb{E}}_{\pi}[C_t B_{t+1} | S_t = s] = \sum_a \pi(a|s) \sum_{s'} p(s', r|s, a) \mathop{\mathbb{E}}_{\pi}[C_t B_{t+1} | _{S_t=s, A_t=a, R_{t+1}=r, S_{t+1}=s'}]. \tag{4.6}$$

Given $(S_t, A_t, R_{t+1}, S_{t+1})$, then $C_t = c(S_t, A_t, R_{t+1}, S_{t+1}) + \epsilon_t$ is effectively constant, and can be taken out of the expectation:

$$\begin{aligned} \mathop{\mathbb{E}}_{\pi,s,a,r,s'}[(c(s,a,r,s') + \epsilon_t) B_{t+1}] &= \mathop{\mathbb{E}}_{\pi,s,a,r,s'}[c(s,a,r,s') + \epsilon_t] \mathop{\mathbb{E}}_{\pi,s,a,r,s'}[B_{t+1}] \\ &= c(s,a,r,s') \mathop{\mathbb{E}}_{\pi,s,a,r,s'}[B_{t+1}]. \end{aligned} \tag{4.7}$$

From the Markov property, we have that $\mathbb{E}_\pi[B_{t+1}|S_t, A_t, R_{t+1}, S_{t+1}] = \mathbb{E}_\pi[B_{t+1}|S_{t+1}]$; furthermore, since $v_\pi(s) = \mathbb{E}_\pi[G_t^\lambda | S_t = s]$ by definition, we have:

$$\mathop{\mathbb{E}}_{\pi}[B_{t+1}|S_{t+1}] = \mathop{\mathbb{E}}_{\pi}[G_{t+1}^\lambda - v_\pi(S_{t+1})|S_{t+1}] = 0.$$

Thus:

$$\mathop{\mathbb{E}}_{\pi}[C_t B_{t+1} | S_t = s] = \sum_a \pi(a|s) \sum_{s'} p(s', r|s, a) c(s, a, r, s') \times 0 = 0. \tag{4.8}$$

Since this holds for arbitrary S_t, the claim follows. ■

With Lemma 4.2 in hand, proving Theorem 4.1 is straightforward:

Proof (Theorem 4.1):

We asserted that:

$$u_\pi(s) \stackrel{\text{def}}{=} \mathop{\mathbb{V}}_{\pi}[G_t^{\lambda,\pi} | S_t = s] = \mathop{\mathbb{E}}_{\pi}\left[(\delta_t^\pi)^2 + \gamma_{t+1}^2 \lambda_{t+1}^2 (G_{t+1}^{\lambda,\pi} - v_\pi(S_{t+1}))^2\right], \tag{4.9}$$

where $G_t^{\lambda,\pi} = R_{t+1} + \gamma_{t+1}(\lambda_{t+1} G_{t+1}^{\lambda,\pi} + (1 - \lambda_{t+1}) v_\pi(S_{t+1}))$ and $\delta_t^\pi = R_{t+1} + \gamma_{t+1} v_\pi(S_{t+1}) - v_\pi(S_t)$.

To show this, we expand from the definition:

$$\begin{aligned}
\mathop{\mathbb{V}}_{\pi}[G_t^{\lambda,\pi}] &= \mathop{\mathbb{E}}_{\pi}[(G_t^{\lambda,\pi} - \mathbb{E}[G_t^{\lambda,\pi}])^2] = \mathop{\mathbb{E}}_{\pi}[(G_t^{\lambda,\pi} - v_\pi(S_t))^2] \\
&= \mathop{\mathbb{E}}_{\pi}\left[(R_{t+1} + \gamma_{t+1}(\lambda_{t+1} G_{t+1}^{\lambda,\pi} + (1 - \lambda_{t+1}) v_\pi(S_{t+1})) - v_\pi(S_t))^2\right] \\
&= \mathop{\mathbb{E}}_{\pi}\left[(R_{t+1} + \gamma_{t+1} v_\pi(S_{t+1}) - v_\pi(S_t) + \gamma_{t+1}\lambda_{t+1}(G_{t+1} - v_\pi(S_{t+1})))^2\right] \\
&= \mathop{\mathbb{E}}_{\pi}\left[(\delta_t^\pi + \gamma_{t+1}\lambda_{t+1}(G_{t+1} - v_\pi(S_{t+1})))^2\right] \\
&= \mathop{\mathbb{E}}_{\pi}\left[(\delta_t^\pi)^2\right] + 2\mathop{\mathbb{E}}_{\pi}\left[\delta_t \gamma_{t+1}(G_{t+1} - v_\pi(S_{t+1}))\right] + \mathop{\mathbb{E}}_{\pi}[\gamma_{t+1}^2 \lambda_{t+1}(G_{t+1} - v_\pi(S_{t+1}))^2].
\end{aligned} \tag{4.10}$$

Applying Lemma 4.2 by identifying C_t with $\gamma_{t+1}\delta_t$, we note that $\mathbb{E}_\pi\left[\delta_t \gamma_{t+1}(G_{t+1} - v_\pi(S_{t+1}))\right] = 0$. Therefore we have:

$$\begin{aligned}
\mathop{\mathbb{V}}_{\pi}[G_t^{\lambda,\pi}] &= \mathop{\mathbb{E}}_{\pi}\left[(\delta_t^\pi)^2\right] + \mathop{\mathbb{E}}_{\pi}[\gamma_{t+1}^2 \lambda_{t+1}^2 (G_{t+1} - v_\pi(S_{t+1}))^2] \\
&= \mathop{\mathbb{E}}_{\pi}\left[(\delta_t^\pi)^2 + \gamma_{t+1}^2 \lambda_{t+1}^2 (G_{t+1}^{\lambda,\pi} - v_\pi(S_{t+1}))^2\right],
\end{aligned} \tag{4.11}$$

as claimed. ■

4.2 Cumulants and Approximation Targets

Manipulating some of the quantities we work with becomes substantially easier if they are expressed in terms of vectors and matrices (see Section 2.2). We describe the notation we will be using, then show how it can be used for the cumulants and their associated returns that we study.

4.2.1 Defining Cumulants

The *Hadamard product*, also known as the Schur product or entrywise product, is a particularly useful tool for expressions involving transition probabilities. For $\mathbf{A}$ and $\mathbf{B}$ both $m \times n$ matrices, their Hadamard product is denoted $\mathbf{A} \circ \mathbf{B}$ and defined:

$$[\mathbf{A} \circ \mathbf{B}]_{ij} \stackrel{\text{def}}{=} [\mathbf{A}]_{ij}[\mathbf{B}]_{ij} = A_{ij}B_{ij}. \tag{4.12}$$

By convention, we denote the elementwise power of a matrix as $\mathbf{A}^{(2)} = \mathbf{A} \circ \mathbf{A}$. When denoting the elementwise power of vectors, we use the exponent without adornment, so $\mathbf{v}^2 = [v_1^2, v_2^2, \dots, v_n^2]^\top$, so as to reserve the parenthesized version for denoting things like the second moment of the return as $\mathbf{v}_\pi^{(2)}$.

Using the Hadamard product, the expected reward $\mathbf{r}_\pi$ is:

$$\begin{aligned}[\mathbf{r}_\pi]_i &= \mathop{\mathbb{E}}_{\pi}[R_{t+1}|S_t = i] = \sum_j p_\pi(i,j)r(i,j) = \sum_j P_{ij}R_{ij} \\ &= [\mathbf{P}_\pi \circ \mathbf{R}]\mathbf{1},\end{aligned} \tag{4.13}$$

where $\mathbf{1} = [1, 1, \dots, 1]^\top$ is the vector of ones, and

$$\begin{aligned}[\mathbf{P}_\pi]_{ij} &\stackrel{\text{def}}{=} p_\pi(i,j), \\ [\mathbf{R}_\pi]_{ij} &\stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{\pi}[R_{t+1}|S_t = i, S_{t+1} = j].\end{aligned} \tag{4.14}$$

That is, $\mathbf{P}_\pi$ is the transition matrix for policy π, and $\mathbf{R}_\pi$ is the reward matrix[3] for that same policy. For remainder of this chapter, we will omit the subscript π on

[3]Note that we tacitly assume that the reward function doesn't depend on the action selected, *e.g.* $r(s,a,s') = r(s,s')$. This simplifies the notation without really affecting the results. We could incorporate action dependence, for example by using tensors or higher order arrays, but this tends towards complication without providing additional insight.

these matrices, since we only consider the on-policy case.

A slightly more complicated example would be the TD error matrix. If we denote by Δ_{ij} the expected TD error for a transition from state i to state j, we observe:

$$\begin{aligned}\Delta_{ij} &= \mathop{\mathbb{E}}_{\pi}[\delta_t | S_t = i, S_{t+1} = j] \\ &= \mathop{\mathbb{E}}_{\pi}[R_{t+1} + \gamma_{t+1} v(S_{t+1}) - v(S_t) | S_t = i, S_{t+1} = j],\end{aligned} \tag{4.15}$$

so the matrix $\mathbf{\Delta}$ can be written:

$$\begin{aligned}\mathbf{\Delta} &\stackrel{\text{def}}{=} \mathbf{R} + \mathbf{1} \otimes (\Gamma \mathbf{v}) - \mathbf{v} \otimes \mathbf{1} \\ &= \mathbf{R} + \mathbf{1}\mathbf{v}_\gamma^\top - \mathbf{v}\mathbf{1}^\top,\end{aligned} \tag{4.16}$$

where we define $\mathbf{v}_\gamma \stackrel{\text{def}}{=} \mathbf{\Gamma v}$, and express the vector outer product using the transpose[4]. The expected TD error vector is then:

$$\begin{aligned}\boldsymbol{\delta} &= (\mathbf{P} \circ \mathbf{\Delta})\mathbf{1} = (\mathbf{P} \circ \mathbf{R})\mathbf{1} + (\mathbf{P} \circ (\mathbf{1}\mathbf{v}_\gamma^\top))\mathbf{1} + (\mathbf{P} \circ (\mathbf{v}\mathbf{1}^\top))\mathbf{1} \\ &= \mathbf{r} + \mathbf{P}\mathbf{\Gamma}\mathbf{v} - \mathbf{v}.\end{aligned} \tag{4.17}$$

The last line follows from the identity $(\mathbf{A} \circ \mathbf{x}\mathbf{y}^\top)\mathbf{z} = \mathbf{x} \circ (\mathbf{A}(\mathbf{y} \circ \mathbf{z}))$. When $\mathbf{v} = \mathbf{v}_\pi = \mathbf{r} + \mathbf{P}\mathbf{\Gamma}\mathbf{v}_\pi$, it's clear that $\boldsymbol{\delta}_\pi = \mathbf{0}$, as expected.

The advantages of this notation become clear when expressing the cumulants and targets for the moments of the return. Here, we focus on expressions for the Monte Carlo return's variance or second moment, although this approach could be modified to yield formulas for the λ-return or the return's higher moments.

The expected squared TD error in state i is:

$$\delta^2(i) \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{\pi}[\delta_t^2 | S_t = i] = \sum_j P_{ij} \Delta_{ij}^2. \tag{4.18}$$

Therefore we define:

$$\boldsymbol{\delta}^{(2)} \stackrel{\text{def}}{=} [\mathbf{P} \circ \mathbf{\Delta}^{(2)}]\mathbf{1}. \tag{4.19}$$

[4]Recall that, for some vectors $\mathbf{a}$ and $\mathbf{b}$ we have $\mathbf{a} \otimes \mathbf{b} = \mathbf{a}\mathbf{b}^\top$ and $[\mathbf{a}\mathbf{b}^\top]_{ij} = a_i b_j$. In particular, we have $[\mathbf{1}\mathbf{a}^\top]_{i,j} = a_j$ and $[\mathbf{a}\mathbf{1}^\top]_{i,j} = a_i$, and .

Similarly, Sobel's variance cumulant (3.14) is, in vector form:

$$\mathbf{m} \stackrel{\text{def}}{=} -\mathbf{v}^2 + \mathbf{P} \circ [\mathbf{R} + \mathbf{1} \otimes (\boldsymbol{\Gamma}\mathbf{v})]^{(2)}\mathbf{1}. \tag{4.20}$$

We write the cumulant for the second moment (3.15) via:

$$\mathbf{r}^{(2)} \stackrel{\text{def}}{=} [(\mathbf{P} \circ \mathbf{R}^{(2)}) + 2(\mathbf{P} \circ \mathbf{R} \circ (\boldsymbol{\Gamma}\mathbf{v})\mathbf{1}^\top)]\mathbf{1}. \tag{4.21}$$

Its expected return (*c.f.* (3.16) is then:

$$\mathbf{v}^{(2)} \stackrel{\text{def}}{=} (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)^{-1}\mathbf{r}^{(2)}. \tag{4.22}$$

The variance targets can therefore be expressed as:

$$\begin{aligned} \mathbf{u}_{\text{SM}} &\stackrel{\text{def}}{=} \mathbf{v}^{(2)} - \mathbf{v}^2, \\ \mathbf{u}_{\text{SOB}} &\stackrel{\text{def}}{=} (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)^{-1}\mathbf{m}, \\ \mathbf{u}_{\text{DV}} &\stackrel{\text{def}}{=} (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)^{-1}\boldsymbol{\delta}^{(2)}, \end{aligned} \tag{4.23}$$

noting that if $\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}$ is invertible, then so is $\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^n$ for $n \geq 1$.

4.3 Equivalence of Expressions for the Variance

For derived quantities where $\mathbf{v} = \mathbf{v}_\pi$, we use the subscript "$\pi$", *e.g.* $\boldsymbol{\Delta}_\pi = \mathbf{r} + \mathbf{P}\boldsymbol{\Gamma}\mathbf{v}_\pi - \mathbf{v}_\pi$ and $\boldsymbol{\delta}_\pi^{(2)} = [(\mathbf{P} \circ \boldsymbol{\Delta}_\pi^{(2)})]\mathbf{1}$. This might seem problematic for the variance of the return, since we have three candidate expressions— which one to use for $\mathbf{u}_\pi$? In fact, they are equivalent when the value function is unbiased, as we will show presently.

Theorem 4.3

Let $\mathbf{v} = \mathbf{v}_\pi$. Then the expressions from (4.23) are equivalent; furthermore they correspond to the variance of the Monte Carlo return. That is,

$$\mathbf{u}_\pi = \underbrace{(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)^{-1}\boldsymbol{\delta}_\pi^{(2)}}_{u_{\text{DV}}} = \underbrace{(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)^{-1}\mathbf{m}_\pi}_{u_{\text{SOB}}} = \underbrace{(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)^{-1}\mathbf{r}_\pi^{(2)} - \mathbf{v}_\pi^2}_{u_{\text{SM}}}, \tag{4.24}$$

where

$$u_\pi(i) = \mathop{\mathbb{E}}_{\pi}[(G_t - v_\pi(S_t))^2 | S_t = i]. \tag{4.25}$$

As part of proving Theorem 4.3, we first establish some results that hold in general, that is, even when $\mathbf{v} \neq \mathbf{v}_\pi$.

Lemma 4.4 Let $\mathbf{m}$ and $\mathbf{r}^{(2)}$ be defined as in (4.20) and (4.21) respectively. Then:

$$\mathbf{m} = \mathbf{r}^{(2)} - (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)\mathbf{v}^2. \tag{4.26}$$

Proof (Proof of Lemma 4.4):

We begin by simplifying $\mathbf{r}^{(2)}$ a bit:

$$\begin{aligned}
\mathbf{r}^{(2)} &= [(\mathbf{P} \circ \mathbf{R}^{(2)}) + 2\mathbf{P} \circ \mathbf{R} \circ (\mathbf{v}_\gamma \mathbf{1}^\top)]\mathbf{1} \\
&= [\mathbf{P} \circ \mathbf{R}^{(2)}]\mathbf{1} + 2[\mathbf{1} \circ (\mathbf{P} \circ \mathbf{R}(\mathbf{1} \circ \mathbf{v}_\gamma))] \\
&= (\mathbf{P} \circ \mathbf{R}^{(2)})\mathbf{1} + 2(\mathbf{P} \circ \mathbf{R})\boldsymbol{\Gamma}\mathbf{v},
\end{aligned} \tag{4.27}$$

where we use the identity $(\mathbf{A} \circ \mathbf{x}\mathbf{y}^\top)\mathbf{z} = \mathbf{x} \circ (\mathbf{A}(\mathbf{y} \circ \mathbf{z}))$. Next, we expand $\mathbf{m}$:

$$\begin{aligned}
\mathbf{m} &= -\mathbf{v}^2 + \mathbf{P} \circ [\mathbf{R} + \mathbf{1}\mathbf{v}_\gamma^\top]^{(2)}\mathbf{1} \\
&= -\mathbf{v}^2 + \underbrace{[\mathbf{P} \circ \mathbf{R}^{(2)}]\mathbf{1} + 2[\mathbf{P} \circ \mathbf{R} \circ (\mathbf{1}\mathbf{v}_\gamma^\top))]\mathbf{1}}_{\mathbf{r}^{(2)}} + [\mathbf{P} \circ (\mathbf{1}\mathbf{v}_\gamma^\top)^{(2)}]\mathbf{1} \\
&= \mathbf{r}^{(2)} - \mathbf{v}^2 + [\mathbf{P} \circ \mathbf{1}(\mathbf{v}_\gamma^2)^\top]\mathbf{1} \\
&= \mathbf{r}^{(2)} - \mathbf{v}^2 + \mathbf{P}\mathbf{v}_\gamma^2,
\end{aligned} \tag{4.28}$$

in which the third line follows from another identity, $(\mathbf{a}\mathbf{c}^\top) \circ (\mathbf{b}\mathbf{d}^\top) = (\mathbf{a} \circ \mathbf{b})(\mathbf{c} \circ \mathbf{d})^\top$. Now, since $\mathbf{v}_\gamma^2 = (\boldsymbol{\Gamma}\mathbf{v})^2 = \boldsymbol{\Gamma}^2\mathbf{v}^2$ since $\boldsymbol{\Gamma}$ is diagonal, we then have:

$$\mathbf{m} = \mathbf{r}^{(2)} - \mathbf{v}^2 + \mathbf{P}\boldsymbol{\Gamma}^2\mathbf{v}^2 = \mathbf{r}^{(2)} - (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)\mathbf{v}^2 \tag{4.29}$$

as claimed. ■

We can also write the DV cumulant in terms of the $\mathbf{m}$ or $\mathbf{r}^{(2)}$ cumulants:

Lemma 4.5 Let $\mathbf{m}$, $\mathbf{r}^{(2)}$, and $\boldsymbol{\delta}^{(2)}$ be defined as in (4.20), (4.21), and (4.19) respectively. Then:

$$\boldsymbol{\delta}^{(2)} = \mathbf{m} - 2\mathbf{v} \circ \boldsymbol{\delta} = \mathbf{r}^{(2)} - (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)\mathbf{v}^2 - \mathbf{v} \circ \boldsymbol{\delta}. \tag{4.30}$$

Proof (Proof of Lemma 4.5):

We start with the definition of $\boldsymbol{\delta}^{(2)}$:

$$\begin{aligned}\boldsymbol{\delta}^{(2)} &= [\mathbf{P} \circ \boldsymbol{\Delta}^{(2)}]\mathbf{1} \\ &= [\mathbf{P} \circ \left(\mathbf{R} + \mathbf{1}\mathbf{v}_\gamma^\top - \mathbf{v}\mathbf{1}^\top\right)^{(2)}]\mathbf{1} \\ &= [\mathbf{P} \circ \left((\mathbf{R} + \mathbf{1}\mathbf{v}_\gamma^\top)^{(2)} + (\mathbf{v}\mathbf{1}^\top)^{(2)} - 2(\mathbf{R} + \mathbf{1}\mathbf{v}_\gamma^\top) \circ (\mathbf{v}\mathbf{1}^\top)\right)]\mathbf{1}.\end{aligned} \tag{4.31}$$

We can extract $\mathbf{m}$ from the above by recognizing

$$[\mathbf{P} \circ \left((\mathbf{R} + \mathbf{1}\mathbf{v}_\gamma^\top)^{(2)}\right)]\mathbf{1} = \mathbf{m} + \mathbf{v}^2. \tag{4.32}$$

Using $(\mathbf{A} \circ \mathbf{x}\mathbf{y}^\top)\mathbf{z} = \mathbf{x} \circ (\mathbf{A}(\mathbf{y} \circ \mathbf{z}))$ we get:

$$[\mathbf{P} \circ \left((\mathbf{R} + \mathbf{1}\mathbf{v}_\gamma^\top) \circ (\mathbf{v}\mathbf{1}^\top)\right)]\mathbf{1} = \mathbf{v} \circ \left(\mathbf{P} \circ (\mathbf{R} + \mathbf{1}\mathbf{v}_\gamma^\top)\right)\mathbf{1}. \tag{4.33}$$

Furthermore, since $[\mathbf{P} \circ (\mathbf{v}\mathbf{1}^\top)^{(2)}]\mathbf{1} = \mathbf{v}^2$ we have

$$\begin{aligned}\boldsymbol{\delta}^{(2)} &= \mathbf{m} + 2\mathbf{v}^2 - 2\mathbf{v} \circ \left(\mathbf{P} \circ (\mathbf{R} + \mathbf{1}\mathbf{v}_\gamma^\top)\right)\mathbf{1} \\ &= \mathbf{m} - 2\mathbf{v} \circ (\mathbf{P} \circ \boldsymbol{\Delta})\mathbf{1} \\ &= \mathbf{m} - 2\mathbf{v} \circ \boldsymbol{\delta},\end{aligned} \tag{4.34}$$

thus proving the claim ■

The preceding results make Theorem 4.3 immediate:

Proof (Proof of Theorem 4.3):

From Lemma 4.4 we have

$$\mathbf{m}_\pi = \mathbf{r}_\pi^{(2)} - (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)\mathbf{v}_\pi^2, \tag{4.35}$$

hence:

$$\begin{aligned}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)^{-1}\mathbf{m}_\pi &= (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)^{-1}\mathbf{r}_\pi^{(2)} - (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)^{-1}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^2)\mathbf{v}_\pi^2 \\ &= \mathbf{v}_\pi^{(2)} - \mathbf{v}_\pi^2,\end{aligned} \tag{4.36}$$

therefore the Sobel and SM expressions are equivalent when $\mathbf{v} = \mathbf{v}_\pi$.

For the DV target, we observe that $\boldsymbol{\delta}_\pi = \mathbf{0}$, so from Lemma 4.5 we have

$$\boldsymbol{\delta}_\pi^{(2)} = \mathbf{m}_\pi - 2\mathbf{v}_\pi \circ \boldsymbol{\delta}_\pi = \mathbf{m}_\pi, \tag{4.37}$$

which implies that the DV and Sobel expressions for $\mathbf{u}_\pi$ are equivalent, and therefore that the SM and DV expressions are also equivalent. ■

4.4 What if the Value Function is Biased?

When $\mathbf{v} \neq \mathbf{v}_\pi$, none of the targets corresponds to the true variance[5]. If the value function is reasonably close to $\mathbf{v}_\pi$, then our approximations for the variance can still be useful[6]. However, if the value function is biased, we have to grapple with the question: what does $\mathbf{u}_{\text{DV}}$ represent?

Some insight can be gained by asking a different question: can we construct a return that expresses the error of our approximate value function? It is easy to show that the difference between our value function and the return corresponds to the discounted sum of TD errors[7]. Let $\mathcal{E}_t \stackrel{\text{def}}{=} G_t - v(S_t)$, and note that it can be expressed recursively:

$$\begin{aligned}\mathcal{E}_t &= G_t - v(S_t) = R_{t+1} + \gamma G_{t+1} - v(S_t) + \gamma(v(S_{t+1}) - v(S_{t+1})) \\ &= \delta_t + \gamma \mathcal{E}_{t+1} = \sum_{n=0}^{\infty} \gamma^n \delta_{t+n}.\end{aligned} \tag{4.38}$$

Taking the expectation, we see:

$$\mathop{\mathbb{E}}_{\pi}[\mathcal{E}_t | S_t = s] = \mathop{\mathbb{E}}_{\pi,s}[G_t] - \mathop{\mathbb{E}}_{\pi,s}[v(S_t)] = v_\pi(s) - v(s). \tag{4.39}$$

So the expected value of $\mathcal{E}_t$ corresponds to the bias, but perhaps more interestingly,

[5] In all cases, the difference between $\mathbf{u}_\pi$ and $\hat{\mathbf{u}}$ will be proportional to $\|\mathbf{v}_\pi - \mathbf{v}\|^2$, as one might guess from the equations.

[6] For many applications, such as stepsize adaptation or other kinds of meta-learning (see White and White (2016), particularly the experiments), so long as the variance estimate is broadly correct it can be an improvement over having no estimate at all.

[7] For easier presentation, we use constant γ, but the argument holds in the GVF setting with state-dependent discounting.

the second moment of $\mathcal{E}_t$ is equivalent to the *Mean Squared Return Error*[8]:

$$\overline{\mathrm{RE}}(\hat{v}) \stackrel{\text{def}}{=} \mathop{\mathbb{E}}_{\pi}[(G_t - \hat{v}(s))^2] = \sum_{s \in \mathcal{S}} d_\pi(s)\, \mathbb{E}[(G_t - \hat{v}(s))^2 | S_t = s]. \tag{4.40}$$

with the obvious equivalence $\mathbb{E}_\pi[\mathcal{E}_t^2] = \overline{\mathrm{RE}}(v)$.

While we can in fact construct a return for a value function's bias, or even its return error, it turns out to not be congenial to approximation. Consider the case where we attempt to estimate the bias using the same features as for learning the value function. For TD(λ) algorithms under linear function approximation, the fixed-point for the weights is:

$$\boldsymbol{\theta}^* = \big[\mathbf{X}^\top \mathbf{D}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}\boldsymbol{\Lambda})^{-1}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma})\mathbf{X}\big]\mathbf{X}^\top \mathbf{D}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}\boldsymbol{\Lambda})^{-1}\mathbf{r}. \tag{2.21}$$

Note that $\mathbf{r} = (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma})\mathbf{v}_\pi$, so we can write:

$$\begin{aligned} \mathbf{v}_{\boldsymbol{\theta}} = \mathbf{X}\boldsymbol{\theta}^* &= \mathbf{X}\big[\mathbf{X}^\top \mathbf{D}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}\boldsymbol{\Lambda})^{-1}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma})\mathbf{X}\big]\mathbf{X}^\top \mathbf{D}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}\boldsymbol{\Lambda})^{-1}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma})\mathbf{v}_\pi \\ &= \mathbf{X}(\mathbf{X}^\top \mathbf{F}_\lambda \mathbf{X}^\top)^{-1}\mathbf{X}^\top \mathbf{F}_\lambda \mathbf{v}_\pi = \boldsymbol{\Pi}_\lambda \mathbf{v}_\pi, \end{aligned} \tag{4.41}$$

where

$$\begin{aligned} \mathbf{F}_\lambda &\stackrel{\text{def}}{=} \mathbf{D}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}\boldsymbol{\Lambda})^{-1}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}) \\ \boldsymbol{\Pi}_\lambda &\stackrel{\text{def}}{=} \mathbf{X}(\mathbf{X}^\top \mathbf{F}_\lambda \mathbf{X}^\top)^{-1}\mathbf{X}^\top \mathbf{F}_\lambda. \end{aligned} \tag{4.42}$$

Which is to say that the TD(λ) fixed-point can be interpreted as a projection of the value function onto the approximation subspace of $\mathbf{X}$ under a weighted L2 norm[9].

If we try to estimate the bias vector $\boldsymbol{\varepsilon} = \mathbf{v}_\pi - \mathbf{v}_{\boldsymbol{\theta}} = (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma})^{-1}\boldsymbol{\delta}$, we see that:

$$\begin{aligned} \hat{\boldsymbol{\varepsilon}}_{\boldsymbol{\theta}} &= \boldsymbol{\Pi}_\lambda \boldsymbol{\varepsilon} = \boldsymbol{\Pi}_\lambda (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma})^{-1}\boldsymbol{\delta} \\ &= \boldsymbol{\Pi}_\lambda (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma})^{-1}(\mathbf{r} + \mathbf{P}\boldsymbol{\Gamma}\mathbf{v}_{\boldsymbol{\theta}} - \mathbf{v}_{\boldsymbol{\theta}}) \\ &= \boldsymbol{\Pi}_\lambda (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma})^{-1}\mathbf{r} + \boldsymbol{\Pi}_\lambda (\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma})^{-1}(\mathbf{P}\boldsymbol{\Gamma} - \mathbf{I})\mathbf{v}_{\boldsymbol{\theta}} \\ &= \boldsymbol{\Pi}_\lambda \mathbf{v}_\pi - \boldsymbol{\Pi}_\lambda \mathbf{v}_{\boldsymbol{\theta}} = \mathbf{v}_{\boldsymbol{\theta}} - \mathbf{v}_{\boldsymbol{\theta}} \\ &= \mathbf{0}. \end{aligned} \tag{4.43}$$

[8] From Sutton and Barto (2018, equation 11.24).

[9] This has been observed before (for example, by Sutton, Szepesvári and Maei (2009), Scherrer (2010) and Maei (2011)) for the case of TD(0) and TD(1), and has presumably been noted in other works.

This makes sense, since at an estimator's fixed-point we would expect that its bias to be minimal[10]. However, it implies that our estimator for $\hat{\boldsymbol{\varepsilon}}_{\boldsymbol{\theta}}$ will be zero for all states, assuming we use the same approximation architecture that produced $\mathbf{v}_{\boldsymbol{\theta}}$.

If we try to construct a target $\hat{\mathcal{E}}_t^{(2)}$ like we did for $\hat{G}_t^{(2)}$ in (3.16), we get:

$$\hat{\mathcal{E}}_t^{(2)} = \underbrace{\delta_t^2 + 2\gamma\delta_t\hat{\varepsilon}(S_{t+1})}_{\text{like } \hat{R}_{t+1}^{(2)}} + \gamma^2\hat{\mathcal{E}}_{t+1}^{(2)}, \tag{4.44}$$

but since $\hat{\varepsilon}_{\boldsymbol{\theta}}(s) = 0$ for $\mathbf{v} = \mathbf{v}_{\boldsymbol{\theta}}$, this just becomes

$$\hat{\mathcal{E}}_t^{(2)} = \delta_t^2 + \gamma^2\hat{\mathcal{E}}_{t+1}^{(2)}, \tag{4.45}$$

which is to say that $\hat{\mathcal{E}}_t^{(2)}$ is equivalent to the δ^2-return at the value function's fixed-point. This comports with the fact that, when $\mathbf{v} = \mathbf{v}_\pi$, $\overline{\text{RE}}$ is entirely comprised by the variance, as we can see from the bias-variance decomposition:

$$\overline{\text{RE}}(v) = \mathop{\mathbb{E}}_{\pi}[(G_t - v(S_t))^2] = \underbrace{\mathop{\mathbb{E}}_{\pi}[(G_t - v_\pi(S_t))^2]}_{\text{variance}} + \underbrace{\mathop{\mathbb{E}}_{\pi}[(v_\pi(S_t) - v(S_t))^2]}_{\text{bias}^2}. \tag{4.46}$$

Circling back to the original question about what the target represents, we could either regard the δ^2-return as a proxy for the variance *or* the mean square return error. But since the learning agent has no way of perceiving its own bias, we prefer to think of $\mathbf{u}_{\text{DV}}$ as representing the variance of the MDP *as experienced by the agent.*

[10]If that were not the case, we could use our estimate of $\hat{\boldsymbol{\varepsilon}}$ to reduce the error further, thereby producing a superior approximation, an apparent contradiction. If $\hat{\boldsymbol{\varepsilon}} \neq \mathbf{0}$, either $\mathbf{v}$ is not at its fixed-point, or we are comparing our approximation against a different objective than the one it is minimizing. This is why TD(0) and TD(1) have different fixed points— they minimize different objective functions (although they have the same solution when $\mathbf{X}$ has rank equal to $|\mathcal{S}|$).

4.5 The Direct Variance TD Algorithm

This section introduces a TD-style algorithm targeting the δ^2-return for variance estimation which we call *Direct Variance Temporal Difference Learning*, abbreviated DVTD: a model-free, online, incremental learning algorithm.

The name is apt for two reasons:

1. It is essentially a temporal-difference algorithm, and enjoys many of the same advantages as standard TD(λ). As with TD(λ), DVTD's updates have time- and memory-complexity of $\mathcal{O}(n)$, where n is the number of parameters.
2. Though similar to VTD in that it uses an estimate of the value function to construct its approximation target, unlike VTD, it does not require further computation to produce the variance— instead, it learns the variance *directly*.

We first present the update equations for DVTD(λ) in the tabular setting:

Tabular DVTD(λ)

$$\left.\begin{aligned} z_t(s) &= \begin{cases} 1 + \gamma_t \lambda_t e_{t-1}(s), & \text{if } s = S_t \\ \gamma_t \lambda_t e_{t-1}(s), & \text{if } s \neq S_t \end{cases} \\ \delta_t &= R_{t+1} + \gamma_{t+1} v(S_{t+1}) - v(S_t) \\ v(S_t) &= v(S_t) + \alpha_t \delta_t e(S_t) \end{aligned}\right\} \text{ TD Component}$$

$$\left.\begin{aligned} \tilde{\gamma}_t &= \gamma_t^2 \kappa_t^2 \\ \tilde{R}_{t+1} &= \delta_t^2 \\ \tilde{z}_t(s) &= \begin{cases} 1 + \tilde{\gamma}_t \tilde{\lambda}_t \tilde{e}_{t-1}(s), & \text{if } s = S_t \\ \tilde{\gamma}_t \tilde{\lambda}_t \tilde{e}_{t-1}(s), & \text{if } s \neq S_t \end{cases} \\ \tilde{\delta}_t &= \tilde{R}_{t+1} + \tilde{\gamma}_{t+1} \tilde{v}(S_{t+1}) - \tilde{v}(S_t) \\ \tilde{v}(S_t) &= \tilde{v}(S_t) + \tilde{\alpha}_t \tilde{\delta}_t \tilde{e}(S_t) \end{aligned}\right\} \text{ DV Component}$$

The TD component estimates the value function and produces the cumulant, δ_t^2, that is then fed into the DV component. The DV component is essentially just another instance of TD(λ) using δ_t^2 in place of R_{t+1}, a different discount factor, $\tilde{\gamma}$, and potentially different bootstrapping and stepsize ($\tilde{\lambda}$ and $\tilde{\alpha}$ respectively).

Here we use $\tilde{\gamma} = \gamma^2\kappa^2$, with different values of κ allowing us to estimate the variance of different λ-returns, such that $\hat{u}(s) \approx \mathbb{E}_\pi[G_t^\kappa]$ For example, with $\kappa = 1$, DVTD estimates the variance of the Monte Carlo return, for $\kappa = 0$ it targets the variance of the one-step TD error, and for $\kappa = \lambda$ it learns the variance of the λ-return used by the TD component.

Schematically, the updates look something like this:

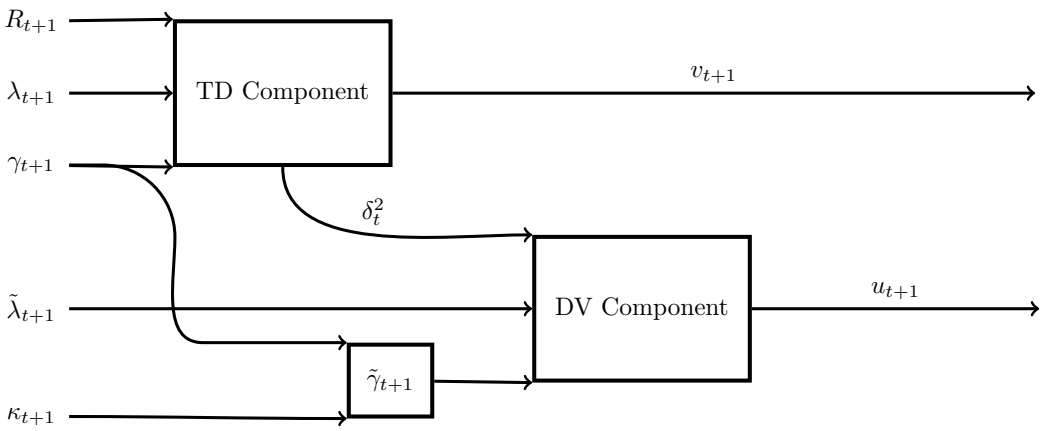

Figure 4.1

Under linear function approximation, the update equations are:

Linear DVTD(λ)

$$\mathbf{z}_t = \mathbf{x}_t + \gamma_t\lambda_t\mathbf{z}_{t-1}$$

$$\delta_t = R_{t+1} + \gamma_{t+1}\boldsymbol{\theta}_t^\top\mathbf{x}_{t+1} - \boldsymbol{\theta}_t^\top\mathbf{x}_t$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t\delta_t\mathbf{z}_t$$

$$\tilde{\gamma}_t = \gamma_t^2\kappa_t^2$$

$$\tilde{R}_{t+1} = \delta_t^2$$

$$\tilde{\mathbf{z}} = \mathbf{x}_t + \tilde{\gamma}_t\tilde{\lambda}_t\tilde{\mathbf{z}}_{t-1}$$

$$\tilde{\delta}_t = \tilde{R}_{t+1} + \tilde{\gamma}_{t+1}\mathbf{w}^\top\mathbf{x}_{t+1} - \mathbf{w}^\top\mathbf{x}_t$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \tilde{\alpha}_t\tilde{\delta}_t\tilde{\mathbf{z}}_t$$

An example implementation of DVTD for linear function approximation is provided in Algorithm 2 (Page 114).

We can observe that it is clearly $\mathcal{O}(n)$ per time-step with respect to the number of features, and requires approximately twice as much time and memory to estimate both the value and variance of the return. Note that, while the basic algorithm

uses TD(λ) for both estimates, we could easily substitute alternate methods for each component— for example, learning the value function with LSTD(λ) and the variance with GTD(λ), or using a different representation for the two components.

The advantage of this simpler formulation is that it make analysis easier. In the next section, we discuss the algorithm's fixed-point and outline a proof of convergence that uses existing results and extends them to DVTD.

4.6 Convergence Results

The ODE method involves comparing a discrete time stochastic approximation scheme (in this case, DVTD) with a related ordinary differential equation, with the goal of showing that:

1. The differential equation converges to some stable equilibrium,
2. The iterates of the approximation scheme become arbitrarily close to the trajectory of the DE.

We described the general framework for this technique in Section 2.6 and now apply it to DVTD. Our proof addresses a modified version of DVTD that makes a number of assumptions, although in our experiments suggest that the algorithm is stable and convergent even without these restrictions.

4.6.1 Setup and Assumptions

In order to apply the ODE method, we first have to establish that DVTD can in fact be written in the required form:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \beta_n[\mathbf{h}(\mathbf{w}_n, \boldsymbol{\theta}_n) + \mathbf{N}_{n+1}], \tag{2.43}$$

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \alpha_n[\mathbf{g}(\mathbf{w}_n, \boldsymbol{\theta}_n) + \mathbf{M}_{n+1}]. \tag{2.44}$$

For $\lambda = \tilde{\lambda} = 0$, DVTD's iterates have the form:

$$\begin{aligned}
\delta_t &= R_{t+1} + \boldsymbol{\theta}_t^\top(\gamma_{t+1}\mathbf{x}_{t+1} - \mathbf{x}_t) \\
\tilde{R}_{t+1} &= \delta_t^2 \\
\tilde{\delta}_t &= \tilde{R}_{t+1} + \mathbf{w}_t^\top(\tilde{\gamma}_{t+1}\mathbf{x}_{t+1} - \mathbf{x}_t) \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha_n \delta_t \mathbf{x}_t \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \beta_t \tilde{\delta}_t \mathbf{x}_t.
\end{aligned} \tag{4.47}$$

If we assume transitions are i.i.d. ($\mathcal{A}$. 2.2), we can condense DVTD's iterates to:

$$\begin{aligned}
\boldsymbol{\theta}_{n+1} &= \boldsymbol{\theta}_n + \alpha_n[r_n + \boldsymbol{\theta}_n^\top(\gamma_n \mathbf{x}'_n - \mathbf{x}_n)]\mathbf{x}_n \\
\mathbf{w}_{n+1} &= \mathbf{w}_n + \beta_n[\tilde{r}_n + \mathbf{w}_n^\top(\tilde{\gamma}_n \mathbf{x}'_n - \mathbf{x}_n)]\mathbf{x}_n,
\end{aligned} \tag{4.48}$$

with $s_k \sim d_\pi(\cdot)$, $s'_k \sim P_\pi(s_k, \cdot)$, $r_k \sim r(s_k, s'_k)$ and $\gamma_k = \gamma(s'_k)$. Then we can see that the TD component fits the mould of (2.44), with

$$\begin{aligned}
\boldsymbol{\theta}_{n+1} &= \boldsymbol{\theta}_n + \alpha_n[\mathbf{g}(\boldsymbol{\theta}_n) + \mathbf{M}_{n+1}] \\
\mathbf{g}(\boldsymbol{\theta}) &= -\mathbf{A}\boldsymbol{\theta} + \mathbf{b} && = -\mathbf{A}[\boldsymbol{\theta} - \boldsymbol{\theta}^*] \\
\mathbf{M}_{n+1} &= r_n + \boldsymbol{\theta}_n^\top(\gamma_n \mathbf{x}'_n - \mathbf{x}_n) - \mathbf{g}(\boldsymbol{\theta}_n) && = \delta_n \mathbf{x}_n - \mathbf{g}(\boldsymbol{\theta}_n),
\end{aligned} \tag{4.49}$$

where

$$\begin{aligned}
\mathbf{A} &= \mathbb{E}_\pi[\mathbf{x}_k(\mathbf{x}_k - \gamma'_k \mathbf{x}'_k)^\top] && = \mathbf{X}^\top \mathbf{D}_\pi(\mathbf{I} - \mathbf{P}_\pi \boldsymbol{\Gamma})\mathbf{X} \\
\mathbf{b} &= \mathbb{E}_\pi[r_k\, \mathbf{x}_k] && = \mathbf{X}^\top \mathbf{D}_\pi \mathbf{r}
\end{aligned} \tag{4.50}$$

are defined as usual[11]. To get $-\mathbf{A}\boldsymbol{\theta} + \mathbf{b} = -\mathbf{A}[\boldsymbol{\theta} - \boldsymbol{\theta}^*]$, we assume that $\mathbf{A}$ is nonsingular[12], so $\boldsymbol{\theta}^* = \mathbf{A}^{-1}\mathbf{b}$ is well-defined, which allows us to substitute $\mathbf{A}\boldsymbol{\theta}^*$ for $\mathbf{b}$. We also note that $\mathbf{g}(\boldsymbol{\theta}, \mathbf{w}) = \mathbf{g}(\boldsymbol{\theta})$ since the DV component does not affect the TD part.

We can identify the analogues of $\mathbf{A}$ and $\mathbf{b}$ in the DV component as well, which should be unsurprising given its roots as a TD method. Taking the expectation for

[11] See Section 2.6.1, particularly the material around (2.35) and (2.36).

[12] This is true even under state-dependent discounting provided that $\mathbf{X}$ is full rank and $\mathbf{P}_\pi$ is an ergodic stochastic matrix. The interested reader can refer to Mahmood 2017, Theorem 32 (and the rest of Chapter 9 if more context is desired).

an arbitrary transition at time k,

$$\mathbb{E}_\pi[(\tilde{r}_k + \mathbf{w}_k^\top(\tilde{\gamma}_k \mathbf{x}'_k - \mathbf{x}_k))\mathbf{x}_k] = \underbrace{\mathbb{E}_\pi[\tilde{r}_k \mathbf{x}_k]}_{\tilde{\mathbf{b}}(\boldsymbol{\theta})} + \underbrace{\mathbb{E}_\pi[(\mathbf{w}_k^\top(\tilde{\gamma}_k \mathbf{x}'_k - \mathbf{x}_k))\mathbf{x}_k]}_{\tilde{\mathbf{A}}}. \tag{4.51}$$

Note that $\tilde{\mathbf{b}}$ is not constant, in contrast to the TD component. It depends on $\boldsymbol{\theta}$ via $\tilde{r}_n = \delta_n^2(\boldsymbol{\theta})$, hence we write $\tilde{\mathbf{b}}(\boldsymbol{\theta})$ to reflect that fact[13]. Conversely, the fact that $\tilde{\mathbf{A}}$ does *not* depend on $\boldsymbol{\theta}$ is also notable. Insofar as we are justified in assuming $\mathbf{A}$ is non-singular, then the same holds for $\tilde{\mathbf{A}}$, provided $\tilde{\gamma}(s) \in [0, 1)$ for all $s \in \mathcal{S}$.

Letting $\tilde{\boldsymbol{\Gamma}}$ be the diagonal matrix with $\tilde{\Gamma}_{ii} = \gamma(i)$, we can rewrite the above in matrix notation:

$$\tilde{\mathbf{A}}(\boldsymbol{\theta}) = \mathbb{E}_\pi[\mathbf{x}_k(\mathbf{x}_k - \gamma_k \mathbf{x}'_k)^\top] \qquad = \mathbf{X}^\top \mathbf{D}_\pi(\mathbf{I} - \mathbf{P}_\pi \tilde{\boldsymbol{\Gamma}})\mathbf{X}, \tag{4.52}$$

$$\tilde{\mathbf{b}}(\boldsymbol{\theta}) = \mathbb{E}_\pi[\tilde{r}_k(\boldsymbol{\theta})\, \mathbf{x}_k] \qquad = \mathbf{X}^\top \mathbf{D}_\pi \tilde{\mathbf{r}}(\boldsymbol{\theta}). \tag{4.53}$$

We can then express the DV component in the desired form:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \beta_n[\mathbf{h}(\boldsymbol{\theta}_n, \mathbf{w}_n) + \mathbf{N}_{n+1}] \tag{4.54}$$

$$\mathbf{h}(\boldsymbol{\theta}, \mathbf{w}) = -\tilde{\mathbf{A}}\mathbf{w} + \tilde{\mathbf{b}}_{\boldsymbol{\theta}} \qquad = -\tilde{\mathbf{A}}[\mathbf{w} - \mathbf{w}^*(\boldsymbol{\theta})] \tag{4.55}$$

$$\mathbf{N}_{n+1} = \tilde{r}_n + \mathbf{w}_n^\top(\tilde{\gamma}_n \mathbf{x}'_n - \mathbf{x}_n) - \mathbf{h}(\boldsymbol{\theta}_n, \mathbf{w}_n) \qquad = \tilde{\delta}_n(\boldsymbol{\theta})\mathbf{x}_n - \mathbf{h}(\boldsymbol{\theta}_n, \mathbf{w}_n), \tag{4.56}$$

where we define $\mathbf{w}^* : \mathbb{R}^d \to \mathbb{R}^d$ as the fixed-point for the DV component for a particular TD component weight vector:

$$\mathbf{w}^*(\boldsymbol{\theta}) \stackrel{\text{def}}{=} \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{b}}(\boldsymbol{\theta}). \tag{4.57}$$

While similar to the iterates for the TD component, the DV part is nonlinear in $\boldsymbol{\theta}$. This makes the analysis more difficult, as $\mathbf{h}(\boldsymbol{\theta}, \mathbf{w})$ is non-Lipschitz, which denies us the ability to simply apply standard theorems for convergence and stability. In order to circumvent those issues, we make an additional assumption in the form of a projection step.

Letting $C \subset \mathbb{R}^d$ be some compact set, chosen such that it is large enough to include

[13] We might also write $\tilde{\mathbf{b}}$ (with the $\boldsymbol{\theta}$ as subscript) as a shorthand.

all possible solutions for the $\boldsymbol{\theta}^*$ and $\mathbf{w}^*$, and define the projection $\boldsymbol{\Upsilon} : \mathbb{R}^d \to C$ as:

$$\boldsymbol{\Upsilon}(\mathbf{z}) \stackrel{\text{def}}{=} \operatorname*{argmin}_{\hat{\mathbf{z}} \in C} \|\mathbf{z} - \hat{\mathbf{z}}\|. \tag{4.58}$$

This obviates issues from possible divergence and in practice actually invoking the projection does not seem to be necessary. While not ideal[14], the use of a projection step is fairly common in the literature.

We end up with updates of the form:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\Upsilon}\Big(\boldsymbol{\theta}_n + \alpha_n[\mathbf{g}(\boldsymbol{\theta}_n) + \mathbf{M}_{n+1}]\Big), \tag{4.59}$$

$$\mathbf{w}_{n+1} = \boldsymbol{\Upsilon}\Big(\mathbf{w}_n + \beta_n[\mathbf{h}(\mathbf{w}_n) + \mathbf{N}_{n+1}]\Big). \tag{4.60}$$

Finally, we also assume that the rewards and features are bounded:

Assumption 4.1 (Bounded Features)

The rewards are bounded with $r_n \leq K_b$ for all $n \geq 0$, as are the features, with $\sup_{s \in \mathcal{S}} \|\mathbf{x}(s)\| \leq K_b$ for some scalar $K_b > 0$.

This assumption is helpful for a number of reasons (as will become evident shortly). It can be weakened somewhat, as we only really need bounds on the moments[15], but that complicates the analysis for dubious benefit.

With the foregoing in mind, we can state our convergence theorem:

Theorem 4.6 (DVTD Convergence for $\boldsymbol{\lambda}$=0)

Let $\{(s_n, s'_n, r_n)\}_{n \geq 0}$ be a sequence of transitions that satisfies $\mathcal{A}$. 2.1 and 2.2, *i.e.*, (s, s', r) is sampled i.i.d., with $r_n = r(s_n, s'_n)$. Assume that the features $\mathbf{x}_n = \mathbf{x}(s_n)$ and rewards r_n are bounded according as in$\mathcal{A}$. 4.1, and that the stepsize sequences $\{\alpha_n\}$ and $\{\beta_n\}$ are chosen such that $\mathcal{A}$. 2.3 holds. Further assume that $\mathbf{A}$ and $\tilde{\mathbf{A}}$ are positive definite matrices.

[14] While we would have preferred to provide a result without recourse to a projection step, the other approaches we tried *also* seemed to entail artificial restrictions, such as mandating a particular stepsize sequence.

[15] For the convergence of TD(0), for example, we need $\mathbb{E}_\pi[\mathbf{x}_n \delta_n] = \mathbb{E}_\pi[\mathbf{x}_n r_n + \mathbf{x}_n(\gamma \mathbf{x}' - \mathbf{x})^\top \boldsymbol{\theta}]$ to be well-defined, which implies that $\mathbf{x}_n$ needs to have a bounded second moment. The DV component complicates this further, as $\mathbf{x}_n \delta_n^2$ shows up in the update equations, which would suggest that the features to be bounded up to their *fifth* moment.

Then for updates of the form (4.59) and (4.60), we have that:

$$\lim_{n\to\infty}(\boldsymbol{\theta}_n, \mathbf{w}_n) = (\boldsymbol{\theta}^*, \mathbf{w}^*) \qquad \text{w.p. 1.} \tag{4.61}$$

Our proof of Theorem 4.6 proceeds by verifying the conditions for Theorem 2.2. Having fitted DVTD to the two-timescale framework, all that remains is to verify the conditions ($\mathcal{A}$. 2.3 to 2.9)that undergird Theorem 2.2. We provide a sketch of our reasoning in the following section, with more detailed exposition in Appendix A.

4.6.2 Proof Sketch

First, we note that some of the conditions are under experimenter control or true based on the assumptions made in Section 4.6.1. For the stepsize requirement ($\mathcal{A}$. 2.3), we assume it is satisfied by appropriate choice of $\{\alpha_n\}$ and $\{\beta_n\}$.

The projection step in (4.59) and (4.60) implicitly guarantees that the iterates are bounded ($\mathcal{A}$. 2.7), and also that the various functions are Lipschitz $\mathcal{A}$. 2.4. For the TD component, we have:

$$\mathbf{g}(\boldsymbol{\theta}) = \mathbf{b} - \mathbf{A}\boldsymbol{\theta} = \mathbf{X}^\top \mathbf{D}\boldsymbol{\delta}(\boldsymbol{\theta}) = -\mathbf{A}[\boldsymbol{\theta} - \boldsymbol{\theta}^*], \tag{4.62}$$

which is clearly linear in $\boldsymbol{\theta}$ and therefore Lipschitz. Showing that the DV component is Lipschitz is more difficult, because while it is linear in $\mathbf{w}$, it is *nonlinear* with respect to $\boldsymbol{\theta}$; however, due to the projection, this issue can be resolved.

To show that $\mathbf{w}^*(\cdot)$ and $\mathbf{h}(\cdot,\cdot)$ are Lipschitz, we first demonstrate that the TD error is bounded, then use that to prove $\boldsymbol{\delta}^{(2)}(\boldsymbol{\theta})$ is bounded. As a bounded and continuous function, $\boldsymbol{\delta}^{(2)}$ is Lipschitz, and therefore so are the other functions, because they are linear with respect to $\boldsymbol{\delta}^{(2)}$.

Convergence of the TD component has been proved in a number of different places[16] which takes care of $\mathcal{A}$. 2.9.

[16] *E.g.* Tsitsiklis and Van Roy 1997 for TD(λ), with finite sample analyses provided in Dalal et al. 2017; Bhandari, Russo and Singal 2018 under more general conditions.

A similar argument can be used to show the fast ODE converges, because with $\boldsymbol{\theta}$ fixed, the DV component is essentially temporal difference learning with a modified reward and discount factor. If the TD component is convergent (which we tacitly assume), we can show that $\mathbf{w}^*(\boldsymbol{\theta}) = \tilde{\mathbf{A}}^{-1}\mathbf{b}(\tilde{\boldsymbol{\theta}})$ exists and is the unique globally asymptotically stable equilibrium point of the fast timescale ODE. Furthermore, due to the projection step it is a Lipschitz function and therefore $\mathcal{A}$. 2.8 is satisfied.

The only remaining assumptions we need to check are the noise bound, $\mathcal{A}$. 2.6, as well as verifying that $\{\mathbf{M}_n\}$ and $\{\mathbf{N}_n\}$ are martingale difference sequences. We can actually show that $\|\mathbf{M}_{n+1}\|$ and $\|\mathbf{N}_{n+1}\|$ are bounded with respect to the weights, which makes establishing bounds on $\mathbb{E}_\pi[\|\mathbf{M}_{n+1}\|^2|\mathcal{F}_n]$ straightforward. It also implies that they are square integrable, and since by construction $\{\mathbf{M}_n\}$ and $\{\mathbf{N}_n\}$ both have expectation zero, we can conclude that they are indeed martingale difference sequences, validating $\mathcal{A}$. 2.5.

Having verified the necessary assumptions, Theorem 2.2 applies, and we can immediately conclude that Theorem 4.6 holds as well.

We note that in practical applications many of these assumptions may be violated to a greater or lesser degree. In our experience, however, we have found DVTD to be well-behaved even without using a projection step or decaying stepsize.

Chapter 5

Experiments

To gather insight into DVTD, we examine its performance on some standard problems and compare it with alternative methods of estimating the variance of the return.

We first consider a discrete MDP, referred to as the *Modified Tamar Chain*, which is a variant of the bounded random walk used by Tamar, Castro and Mannor 2016. For that series of experiments we compare the moment-based DVTD and VTD, but also report on the performance of the C51 distributional reinforcement learning algorithm. Later, we test DVTD and VTD in the more complicated Mountain Car testbed. We performed experiments in a number of different environments, but found that the outcomes were broadly similar; to avoid undue repetition we content ourselves with reporting on these two illustrative cases.

Pseudocode for the algorithms used is given in Appendix C. The Python code used in the actual experiments is also available online[1].

5.1 Modified Tamar Chain

Random walk MDPs are fairly common in the stochastic approximation literature. The MDP we analyze here was used in (Tamar, Castro and Mannor 2016), wherein they estimate the variance via the second moment, making it an appropriate testbed for comparison. We use a version with 11 states, as opposed to the original 32 states, because the additional states made it harder to report on and visualize without qualitatively affecting the results.

[1] https://github.com/rldotai/varcompfa

The environment consists of a chain of states numbered 0 to 10, with the agent starting in state 0 for all episodes. Each episode terminates when the agent reaches state 10. In each non-terminal state i, the agent has a probability of transitioning to state $i+1$ ("right") with probability 0.7, and to state $i-1$ ("left") with probability 0.3, except in state 0, where if the agent moves to the left it remains in the same state. For each transition to a non-terminal state, the agent receives a reward of -1; when transitioning to the terminal state it receives a reward of 0. We used a constant discount factor of 1, meaning that each state's value essentially represents the expected time until termination.

An illustration of the MDP is provided in Fig. 5.1.

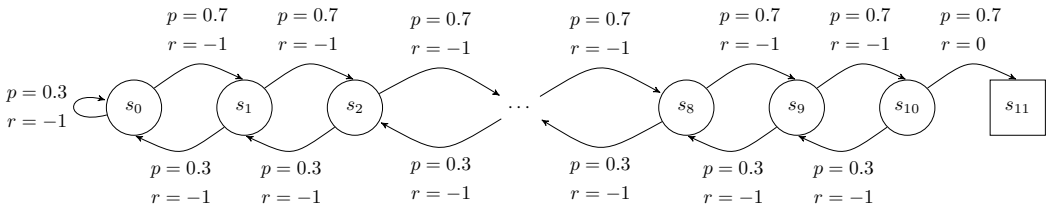

Figure 5.1: *A modified version of an MDP from Tamar, Castro and Mannor (2016); essentially it is a biased random walk. The agent starts each episode in state* 0*, with the episode terminating once the agent reaches state* 11*. Its transition probabilities are biased so that it tends to transition to the "right", towards the terminal state.*

5.1.1 Methodology

We conducted experiments to compare the algorithms across various hyperparameter settings.

One way of evaluating a learning agent's performance is by comparing its approximation with the actual function we seek to estimate. For value functions, this is frequently the standard *Mean Square Value Error*, which measures the distance between the learned value function and v_π. With $d_\pi : \mathcal{S} \to \mathbb{R}$ the policy's stationary distribution, the value error is defined as[2]:

$$\overline{\text{VE}}(\hat{v}) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} d_\pi(s)[(v_\pi(s) - \hat{v}(s))^2] = \mathbb{E}_\pi[(v_\pi(s) - \hat{v}(s))^2]. \tag{5.1}$$

[2]From Sutton and Barto (2018, Equation 9.1), although we are using the on-policy version of $\overline{\text{VE}}$. That is to say, the error is weighted with respect to the on-policy distribution; other weightings are sometimes considered (*e.g.*, the uniform distribution) but this is arguably the more natural choice.

The advantage of $\overline{\text{VE}}$ is that it measures the difference between $\hat{v}$ and v_π absent noise or other distractions, and its minimum is zero, achieved when $\hat{v} = v_\pi$. Its main drawback is that we have to be able to calculate v_π and d_π, although this is fairly straightforward for discrete MDPs.

We can also define analogous errors[3] for the second moment and variance:

$$\overline{\text{VE}}(\hat{v}^{(2)}) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} d_\pi(s)[(v_\pi^{(2)}(s) - \hat{v}^{(2)}(s))^2], \tag{5.2a}$$

$$\overline{\text{VE}}(\hat{u}) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} d_\pi(s)[(u_\pi(s) - \hat{u}(s))^2]. \tag{5.2b}$$

For scaling reasons, we tend to report the *Root Mean Square Value Error*, abbreviated RMSVE, which is simply:

$$\text{RMSVE}(\hat{v}) \stackrel{\text{def}}{=} \left(\sum_{s \in \mathcal{S}} d_\pi(s)[(v_\pi(s) - \hat{v}(s))^2] \right)^{1/2} = \sqrt{\overline{\text{VE}}(\hat{v})}. \tag{5.3}$$

We plot the return's expected value, second moment, and variance for the Tamar Chain in Fig. 5.2.

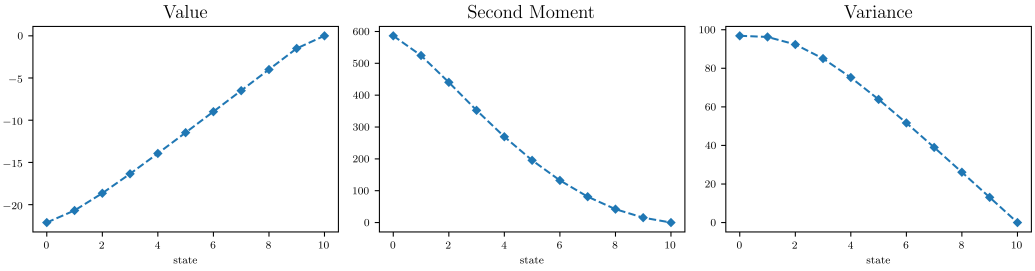

Figure 5.2: *The expected value, second moment, and variance of the return for the Modified Tamar Chain problem versus state. The value function is roughly linear, except at the ends of the chain, while the second moment is nonlinear. The variance is also roughly linear, except near the starting states.*

Representation We performed experiments with two different state representations. The first representation, referred to as *tabular features*[4], with each state's

[3] Which we also refer to as "value errors", since VTD and DVTD both assign a value to states, and the formula for those errors is the same as for the standard $\overline{\text{VE}}$ but using $v_\pi^{(2)}$ and u_π used in place v_π as the target function.

[4] Because each feature can be thought of as an entry in a table, one for each of the environment's states.

feature vector being given as:

$$[\mathbf{x}(i)]_j = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases} \qquad \text{for } i \in \mathcal{S}. \tag{5.4}$$

For example, the feature vector for state $s = 1$ is given by $\mathbf{x}(1) = [0, 1, 0, \ldots, 0, 0]^\top$.

The second representation, which we call *Modified Tamar Features*, uses a rescaled version of the features provided by Tamar, Castro and Mannor (2016). The original representation consisted of two sets of features, one for the value estimator, $\tilde{\mathbf{x}}_{\text{val}}(s) = [1, s]^\top$, and another for the variance estimator, $\tilde{\mathbf{x}}_{\text{var}}(s) = [1, s, s^2]^\top$. We found those features could produce instability, since the relative magnitudes of the features for each state can vary widely. For example, with $\tilde{\mathbf{x}}_{\text{var}}$ the largest feature vector has a norm two orders of magnitude greater than that of the smallest[5].

To address this, we rescaled the features while keeping the general scheme:

$$\mathbf{x}_{\text{val}}(s) \stackrel{\text{def}}{=} \left[1, \frac{(s+1)}{|\mathcal{S}|}\right]^\top = \left[1, \frac{(s+1)}{11}\right]^\top, \tag{5.5a}$$

$$\mathbf{x}_{\text{var}}(s) \stackrel{\text{def}}{=} \left[1, \frac{(s+1)}{|\mathcal{S}|}, \frac{(s+1)^2}{|\mathcal{S}|^2}\right]^\top = \left[1, \frac{(s+1)}{|\mathcal{S}|}, \frac{(s+1)^2}{121}\right]^\top. \tag{5.5b}$$

As the modified features are only subject to a linear transformation, the algorithms' fixed points for $\mathbf{v}$ and $\mathbf{u}$ were unchanged, but the rescaling permits a wider choice of stepsize and removes a potential source of numerical error.

5.1.2 Tabular Experiments

In the tabular case, the estimators can represent the return's expected value, second moment, and variance exactly. It is the simplest setting available, and helps verify that our implementation works correctly. The tabular setting is also useful when examining the algorithms under function approximation: if we discover unexpected or undesirable behaviour in the FA case, then by comparing with the tabular results we can determine whether the fault lies with the algorithms themselves or is an artifact of the representation we used.

[5]That is, $\|\tilde{\mathbf{x}}(0)\| = 1$, while $\|\tilde{\mathbf{x}}(10)\| \approx 100$

We ran the experiment twelve times, with each run consisting of 125,000 time-steps, using a constant stepsize of $\alpha = 0.01$ for all time-steps.

When $\lambda = \bar{\lambda} = 0$, DVTD and VTD appear to be equivalent, producing very similar estimates at each point in time, with similar error curves. However, for other choices of λ or $\bar{\lambda}$, the transient estimates are observably different, with noticeably different RMSVE curves, as can be seen in Fig. 5.3.

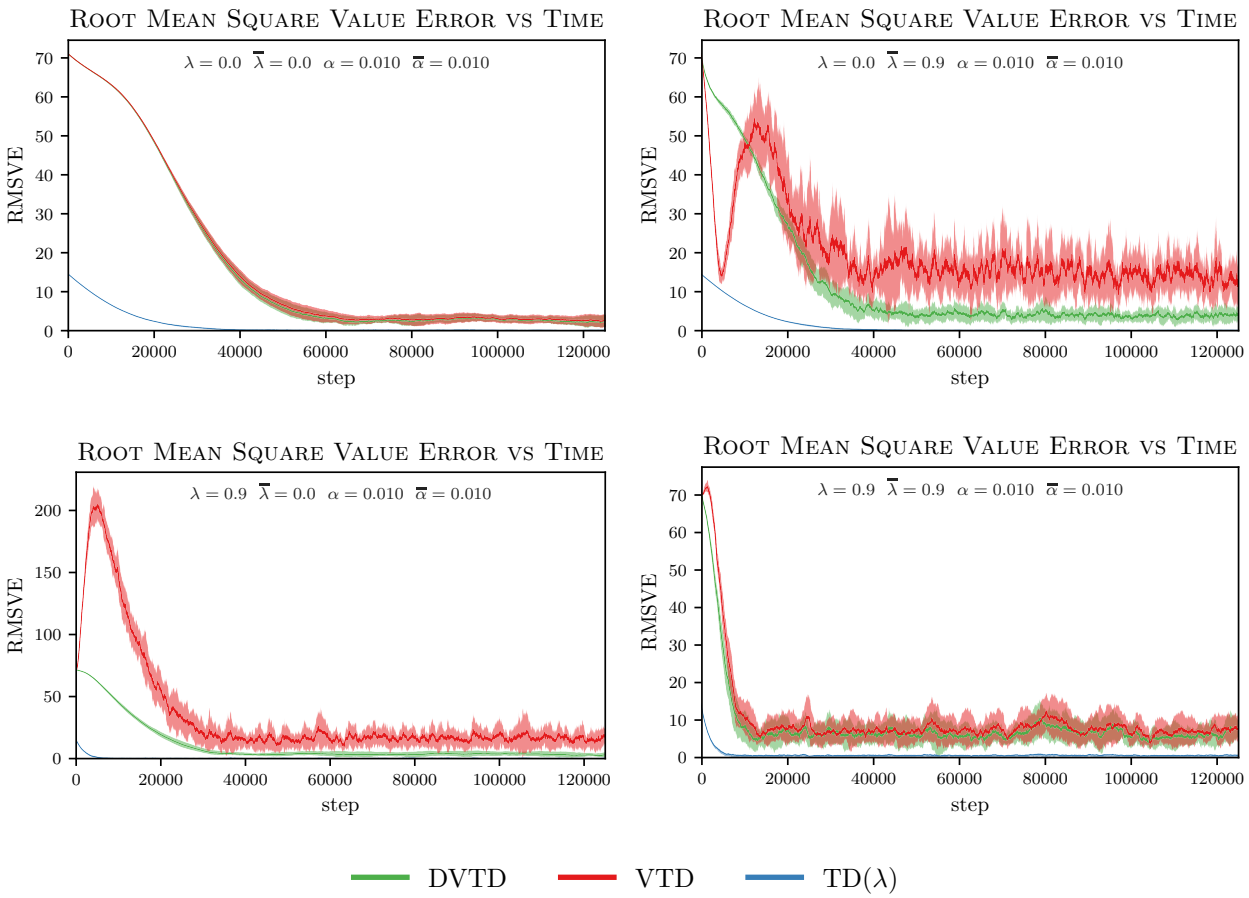

Figure 5.3: *RMSVE for different choices of λ and $\bar{\lambda}$, with the same stepsize. The top left has $\lambda = \bar{\lambda} = 0$; top right $\lambda = 0$ and $\bar{\lambda} = 0.9$; bottom left $\lambda = 0.9$ and $\bar{\lambda} = 0.0$; bottom right $\lambda = \bar{\lambda} = 0.9$. The average error is plotted as a solid line, while shaded regions represent the square root of the variance.*

In the table-lookup setting, all the algorithms we use should be able to represent their targets perfectly, although with constant stepsize there will typically be a certain amount of steady-state error due to noise. Here, nonzero values of λ or $\bar{\lambda}$ are somewhat counterproductive because they entail greater variance without reducing the bias. Indeed, we observe that the lowest asymptotic error is achieved when

$\lambda = \bar{\lambda} = 0$.

TD(λ) exhibits a roughly exponential convergence rate to its asymptote, which was accelerated when using higher values of λ, whereas DVTD and VTD seem to have roughly two phases to their convergence. At the beginning of each experiment, their error either decreased slowly or (in VTD's case) had a transient spike[6], before eventually heading towards their respective asymptotes. We observe that these phases seemed to correspond to some threshold level of accuracy for the TD's value estimate. Once the value estimate was sufficiently close to v_π, they both began to approach their fixed-point at a faster rate[7].

Both DVTD and VTD converge faster for higher values of λ and $\bar{\lambda}$, albeit with higher asymptotic noise. This is also the case for TD(λ), but the additional noise from higher λ is almost unnoticeable in comparison. Finally, we note that across all the experiments we performed, DVTD had lower asymptotic error, faster convergence, and less noise relative to VTD.

5.1.3 Function Approximation

Next, we consider the algorithms under function approximation. Using the Modified Tamar Features from (5.5), the value function cannot be perfectly captured[8]. We first examine the algorithms' behaviour analytically, and then compare how they perform in simulated experiments.

Analytical Results

If the value function is not exact, the approximation target for DVTD is no longer the true variance of the return, nor is VTD's target the true second moment. This problem is compounded by the fact that such a target may *also* be imperfectly representable with the given features, so we are effectively making an approximation

[6]Intriguingly, VTD only seemed to experience high transient error when $\lambda \neq \bar{\lambda}$, and it seemed to perform best when $\lambda = \bar{\lambda}$. We note that this also holds for the other settings of λ and $\bar{\lambda}$ that we tried, but further experiments are needed to gauge whether this is mere coincidence rather than because of some underlying principle.

[7]This is particularly noticeable for $\lambda = \bar{\lambda} = 0$, where the error curves are roughly sigmoidal.

[8]Because (5.5a) is a linear function of the state number, while the the expected return for the MDP is nonlinear (see Fig. 5.2).

from an approximation. Furthermore, the approximation targets and the algorithms' fixed-points are impacted by the bootstrapping hyperparameters λ and $\bar{\lambda}$.

Performing a full hyperparameter sweep can be quite time consuming if done via simulated experiments. Fortunately, using the methods described in Section 2.5, we can compute the required quantities analytically, which provides a straightforward way of evaluating the fixed-points of the algorithms.

For each value of λ, we computed the value function from TD(λ)'s fixed-point, and used that to compute the approximation targets $\mathbb{E}_\pi[H_t]$ and $\mathbb{E}_\pi[\hat{G}_t^2]$. We then solved for DVTD and VTD's respective fixed-points for different choices of $\bar{\lambda}$. Altogether, we sampled $(\lambda, \bar{\lambda}) = \{0.0, 0.05, \dots, 1.0\} \times \{0.0, 0.05, \dots, 1.0\}$, for a total of 441 different points. We plot the results in Fig. 5.4.

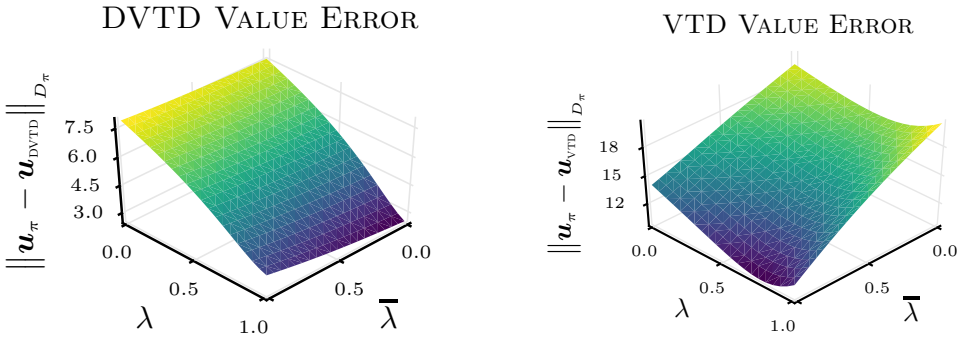

Figure 5.4: *The RMSVE for the fixed-points of DVTD (left) and VTD (right) on the Tamar Chain task as a function of λ and $\bar{\lambda}$ using the Modified Tamar Features.*

The most noteworthy observation is that DVTD's fixed-point has lower RMSVE than VTD, even when pitting VTD's best hyperparameter settings against DVTD's worst.

The behavior of the fixed-points' RMSVE as a function of bootstrapping was somewhat unexpected. We predicted the accuracy of the estimates would improve with higher λ or $\bar{\lambda}$, but we instead observe that DVTD's error increases slightly as a function of $\bar{\lambda}$; VTD's RMSVE decreases with higher $\bar{\lambda}$, but its minimum when λ is at $\lambda = 0.8$ rather than at $\lambda = 1$.

Simulation Experiments

While analytical results can be informative, we also seek to evaluate how DVTD performs with real data.

1. Does it converge to the predicted fixed-points?
2. How does it behave under varied hyperparameters?
3. Is it robust and stable enough to be used in practice?

To answer these questions, we performed a number of experiments using different bootstrapping (λ and $\bar{\lambda}$) as well as varied stepsizes (α and $\bar{\alpha}$). For comparative purposes, we include VTD and C51 as alternative variance estimation methods.

The typical experiment has error curves like that of Fig. 5.5. As in the earlier tabular experiments (Section 5.1.2), DVTD and VTD exhibited two-phase convergence, meandering until the value estimate (from TD(λ)) reached a certain level of accuracy, at which point their RMSVE begins to decrease. Interestingly, the RMSVE for C51's value estimates *also* flattened out well before its variance estimation error. During the transient and steady-state periods, VTD had significantly higher noise than the other algorithms; C51's estimates were more consistent, albeit at the price of much slower convergence.

To provide additional context, in Fig. 5.6, we plot the RMSVE curves for different values of λ and $\bar{\lambda}$ with constant stepsize. Overall, we found that DVTD outperforms VTD and C51 in terms of asymptotic RMSVE. This is somewhat expected from the analytical results— DVTD's fixed-point has the lowest RMSVE of all the variance estimators, so insofar as it converges (and we haven't made a mistake in our calculations), then it should have the best performance.

For the moment-based algorithms (DVTD and VTD), the increased accuracy of $\hat{v}$ from higher values of λ was generally not worth the additional noise incurred. Both DVTD and VTD appear to magnify the effects of variance in the value function, to the point that their asymptotic error seems to hover above their predicted fixed-

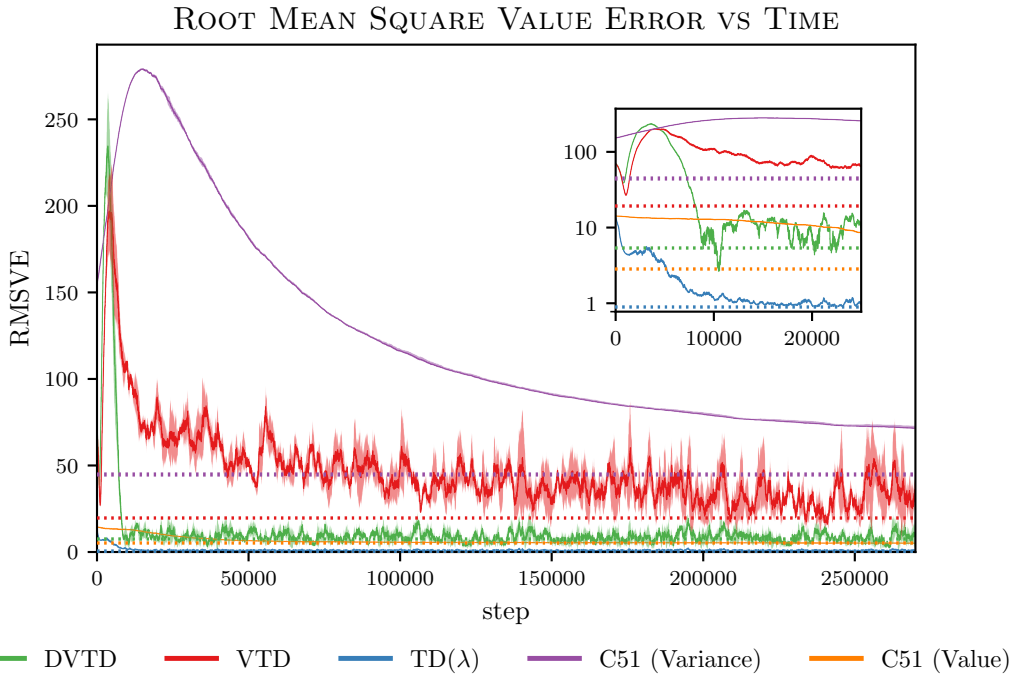

Figure 5.5: *A plot of the RMSVE for the TD, DVTD, VTD, and C51 algorithms vs time in the Tamar Chain environment under function approximation. Inset is a log-scale plot of the starting time-steps. The analytical fixed-points for each algorithm are shown as dotted lines.*

points[9]. We can observe that for $\lambda = 0$, their error curves hover around their predicted fixed points; higher values have better asymptotic errors but these are not easily reached. On the other hand, C51 benefits from higher values of λ without exhibiting much increase in variance.

While we know (from the earlier analysis) that DVTD's fixed-point degrades slightly with higher $\bar{\lambda}$, our experiments revealed that this was a minor problem compared to the effect of the greater noise. Increasing $\bar{\lambda}$ did accelerate convergence, but only once the value estimate was close enough to its asymptote for the approximation targets to be stable. For every choice of bootstrapping or stepsize, DVTD exhibited lower variance relative VTD. When comparing DVTD and VTD, we conjecture that this will hold for arbitrary hyperparameter settings, so long as VTD and DVTD share the same hyperparameters, since it has been observed to be the

[9]This is understandable in light of the fact that they are using TD's value function, which for constant stepsize will be of the form $\hat{v} = v_{\text{TD}} + \textit{noise}$. The expressions for both algorithms' fixed-points include a term for the squared value function, meaning that the effect of the noise does not average out. In experiments with decreasing stepsize, we observed that both DVTD and VTD do in fact converge to their analytic fixed-points.

case in every experiment we performed.

In contrast, C51 produced remarkably consistent estimates, even with higher $\bar{\alpha}$ and λ. This fact, along with the slower convergence for C51 relative to VTD and DVTD when using the same stepsize, suggested that a straightforward comparison across different choices of $\bar{\alpha}$ might not be appropriate. Even for very high stepsize ($\bar{\alpha} = 0.5$), C51 was very stable and appeared to converge to its fixed-point, while VTD and DVTD's asymptotes were noticeably above their predicted fixed-point. We can see this in Fig. 5.7 and as part of the montage in Fig. 5.9.

While C51's stability at higher stepsizes was impressive, we note that it nonetheless failed to converge before DVTD, and even with $\bar{\alpha} = 0.5$, C51's variance estimate still takes longer to converge than DVTD does with $\alpha = \bar{\alpha} = 0.015625$ (shown in Fig. 5.8); at such low stepsizes, DVTD exhibits comparable stability.

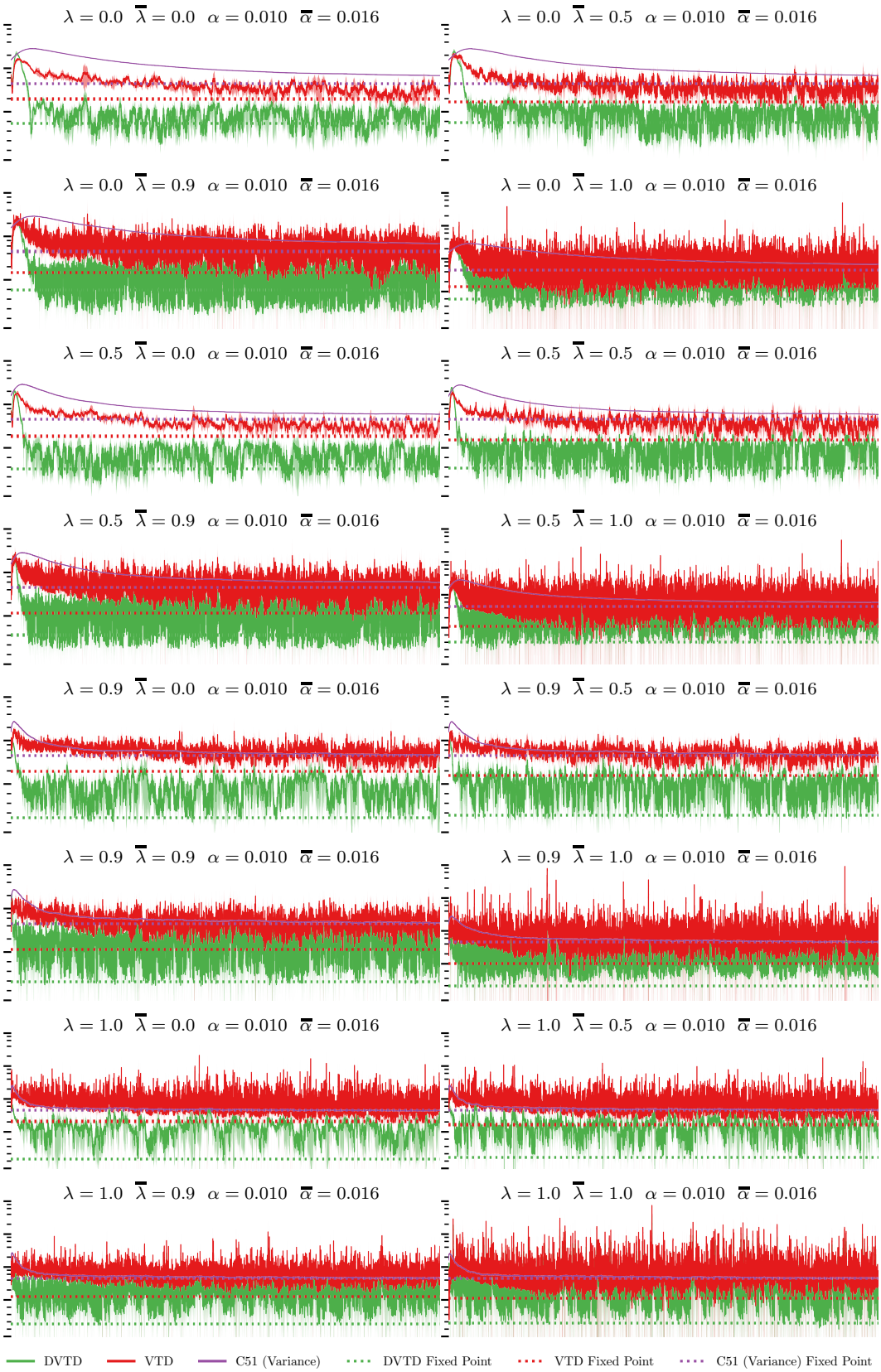

Figure 5.6: *Simplified plots of the* RMSVE *for DVTD, VTD, and C51 in the Tamar Chain task under function approximation with various hyperparameters. The y-axis is linear until* $y = 10$*, after which it has a logarithmic scale. Each algorithm's fixed-point as determined analytically is shown with a dotted line.*

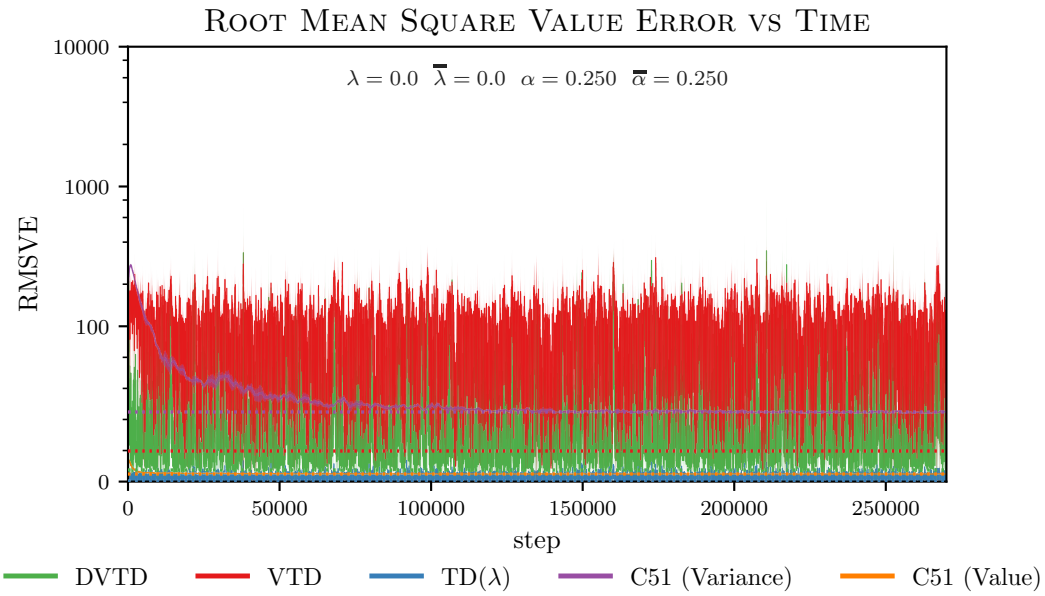

Figure 5.7: *A plot of the RMSVE for the TD, DVTD, VTD, and C51 algorithms over time in the Tamar chain environment under function approximation.*

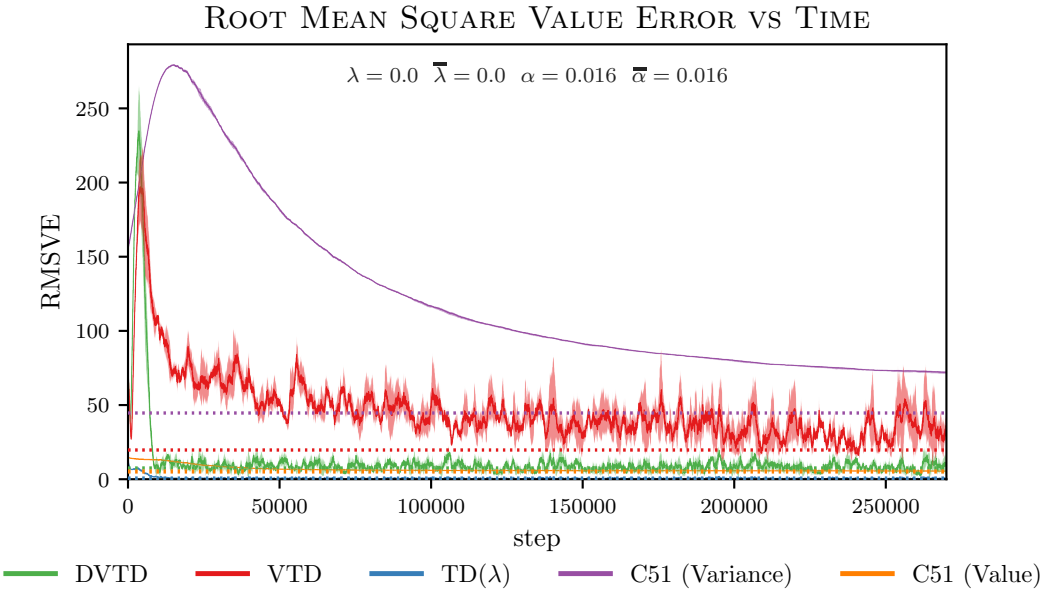

Figure 5.8: *A plot of the RMSVE for the TD, DVTD, VTD, and C51 algorithms over time in the Tamar chain environment under function approximation.*

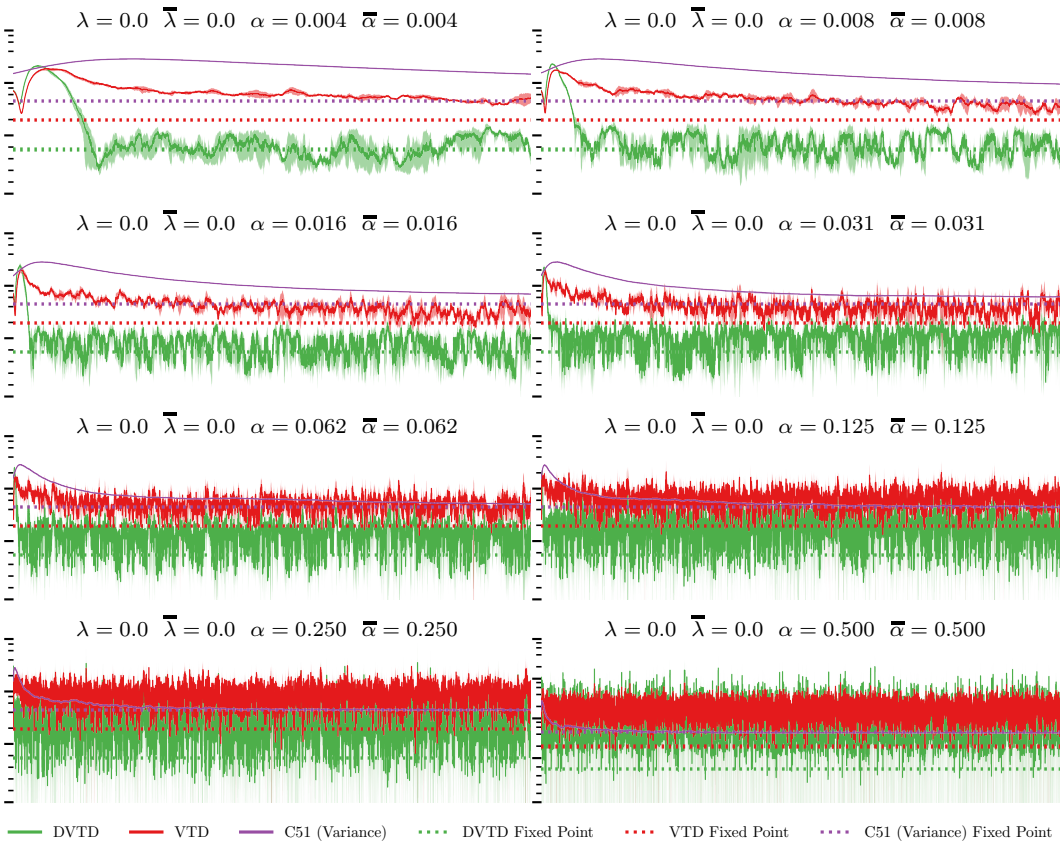

Figure 5.9: *Simplified plots of the* RMSVE *for DVTD, VTD, and C51 in the Tamar Chain environment with function approximation, showing the effects of varying stepsize on the algorithms' performance. The y-axis is linear until $y = 10$, after which it uses a logarithmic scale. Each algorithm's fixed point is shown with a dotted line.*

5.1.4 Summary

We found that all algorithms were capable of producing reasonable estimates for the variance of the return, with DVTD's fixed-point having the lowest RMSVE, followed by VTD and then C51. From the analytical results we observed that the accuracy of the value function strongly determines how good the approximation targets are, yet in practice any benefit from a better $\hat{v}$ tends to be overshadowed by the increased variance from higher λ. With suitably chosen stepsize parameters, all algorithms were capable of reaching their predicted fixed-points.

In general, DVTD performed at least as well and usually better relative to VTD for every choice of hyperparameters we investigated, in terms of convergence rate and stability, in both the tabular setting and under function approximation. Both moment-based variance learning algorithms had two phases to their convergence, appearing to begin an abrupt descent in their error curves once the value function was sufficiently accurate but before it reached its asymptote. We also observed that C51's value estimate converged before its variance estimate, although without the two-phased behavior seen with DVTD and VTD. Notably, C51 was much more stable compared to the moment-based variance approximation algorithms, although at the price of much slower convergence.

Insofar as we are interested in learning the variance of the return, DVTD appears to be clearly preferable over VTD and C51, although further experiments in other domains would be needed to establish if this is consistently true.

5.2 Mountain Car

The mountain car problem (Singh and Sutton 1996) is a standard reinforcement learning test domain, which models an underpowered car as it attempts to climb a hill from a starting point at the base of a valley. The car cannot simply drive up the hill from its initial position due to its limited acceleration. Instead, it has to build up speed by rolling some distance up one hill, and then reversing direction towards the other hill. Each reversal increases the height to which it can climb before it runs out of speed, until eventually it has built up enough energy to reach the summit. In this domain, the desired control policy one that can reach the summit in the fewest time steps.

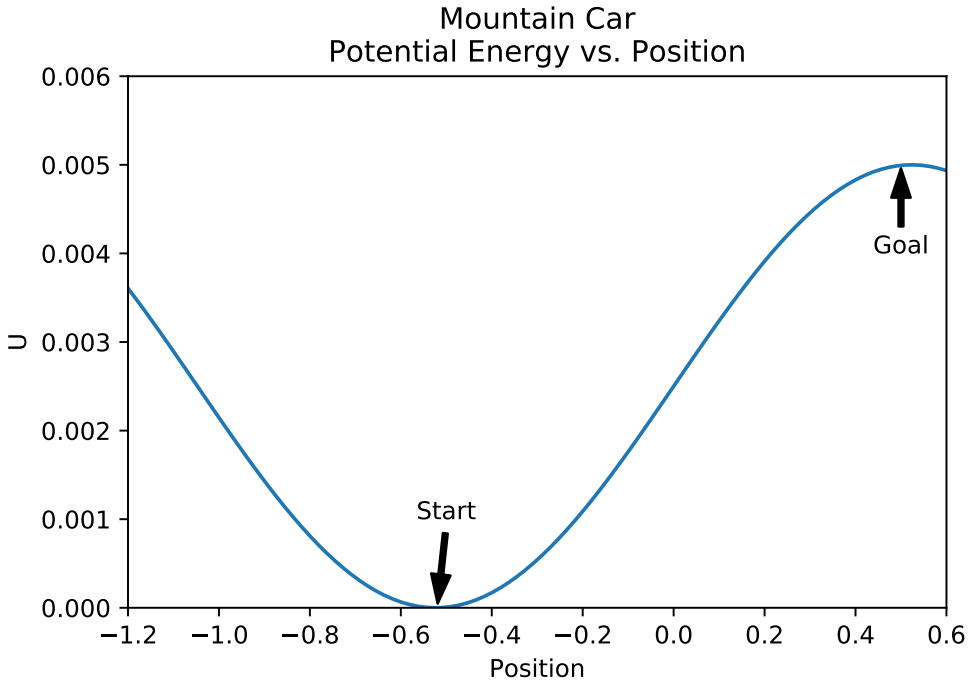

Figure 5.10: *Here we show the car's potential energy as a function of position. This domain is essentially a physics problem where the agent needs to gather sufficient energy to reach the summit. The agent can accelerate forwards, backwards or idle; as depicted, "forwards" means to the right. It does not have enough acceleration to drive directly to the goal starting from rest at the hill's base, but the problem can be solved by rocking back and forth to build up speed.*

Mountain Car was chosen as a test case because the dynamics are deterministic, yet function approximation is needed to account for the continuous state space. In a sense, the "true" variance in this problem is zero (assuming the policy is fixed and deterministic), but from the agent's perspective, it will appear to be stochastic so

long as the representation is imperfect.

The state space represents the car's position and velocity, $\mathbf{s}_t = (x_t, v_t) \in \mathcal{S} \subseteq \mathbb{R}^2$. The goal is reached when the agent's position is greater than 0.5, regardless of its velocity. For all non-terminal steps the reward is -1. At each time step t, the agent can choose to accelerate backwards, idle, or accelerate forwards; the action space is then $a_t \in \{0, 1, 2\} = \mathcal{A}$.

The car has a maximum velocity of ± 0.07, and its position is bounded by $[-1.2, 0.6]$. When it would exceed these limits, its position/velocity is clipped to keep it in bounds, except if it reaches the minimum position with negative velocity ($x \leq -1.2, v < 0$), its velocity is set to zero and its position is set to -1.2.

The preceding verbiage can be summarized via the following update equations:

$$\begin{array}{lll} \alpha_{\text{car}} = 0.001 & \bar{v}_{t+1} = \alpha_{\text{acc}}(a_t - 1) + \alpha_{\text{grav}} \cos(3x_t) & \bar{x}_{t+1} = x_t + \tilde{v}_{+1} \\ \alpha_{\text{grav}} = -0.0025 & \tilde{v}_{t+1} = \text{clip}[\bar{v}_{t+1}, -0.07, 0.07] & x_{t+1} = \text{clip}[\bar{x}_{t+1}, -1.2, 0.6] \\ & v_{t+1} = 0 \text{ if } x_{t+1} = -1.2, \text{ else } \tilde{v}_{t+1} & \end{array} \tag{5.6}$$

5.2.1 Experiment Setup

For our experiments, we first trained a control agent to solve the task starting from a variety of points within the state space, using the implementation of Mountain Car provided by OpenAI Gym (Brockman et al. 2016).

The control agent was Q(λ), which was selected because it is fast to train and deterministic when selecting actions greedily[10]. We used a discount parameter of $\gamma_t = 0.9999$ for all non-terminal t, which reflects the task's objective of training the agent to reach the summit in the minimum time. Exploration was incentivized using optimistic initialization and ε-greedy action selection[11], with $\epsilon = 0.002$. The learning rate was a constant $\alpha = 0.01$, and we chose $\lambda = 0.5$ somewhat arbitrarily.

[10] Other algorithms or different parameterizations might be better at solving the task, but since we only sought to generate a dataset on which we could test DVTD and VTD, Q(λ) was sufficient.

[11] *E.g.*, for $\epsilon = 0.0$ the action is chosen to maximize the state-action value, but for $\epsilon = 1/3$ the agent would pick a random action a third of the time.

The features used were normalized radial basis functions (NRBF), which map the continuous, low-dimensional state space into a higher dimensional (but much sparser) feature vector[12].

A radial basis function (RBF) is a function of the form:

$$\mathrm{RBF}(\mathbf{s}) = \exp\left\{-(\mathbf{s}-\boldsymbol{\eta})^\top \boldsymbol{\Sigma}(\mathbf{s}-\boldsymbol{\eta})\right\}, \tag{5.7}$$

where $\mathbf{s}, \boldsymbol{\eta} \in \mathcal{S} \subseteq \mathbb{R}^n$ and $\boldsymbol{\Sigma} \in \mathbb{R}_+^{n\times n}$. The parameter $\boldsymbol{\eta}$ controls where the function is centered (*i.e.*, has the highest activation), and $\boldsymbol{\Sigma}$ determines the shape of the RBF.

The "normalized" aspect of NRBFs comes from the fact that after computing the activations of the individual RBFs, we rescale their values such that they sum to unity. Given radial basis functions $\{f_1, \ldots, f_d\}$, with each f_i having its own corresponding $\boldsymbol{\eta}_i$ and $\boldsymbol{\Sigma}_i$, the i-th NRBF is given by:

$$\bar{f}_i(\mathbf{s}) = \mathrm{NRBF}_i(\mathbf{s}) = \frac{f_i(\mathbf{s})}{\sum_j f_j(\mathbf{s})} = \frac{e^{-(\mathbf{s}-\boldsymbol{\eta}_i)^\top \boldsymbol{\Sigma}_i(\mathbf{s}-\boldsymbol{\eta}_i)}}{\sum_j e^{-(\mathbf{s}-\boldsymbol{\eta}_j)^\top \boldsymbol{\Sigma}_j(\mathbf{s}-\boldsymbol{\eta}_j)}}. \tag{5.8}$$

We chose the centers of the RBFs along a uniform 30×30 grid over the state space, for a total of 900 features. As such, care had to be taken with $\boldsymbol{\Sigma}$ to ensure the features have appropriate scale. Mountain Car's state space is $[-1.2, 0.6] \times [-0.07, 0.07]$, meaning that the the basis functions are somewhat closer for the velocities compared to the positions. A rule of thumb (Kretchmar and Anderson 1997) suggests that the scaling should be proportional to I_i^2, where I_i is the size of the interval for dimension i of the state space. Employing this rule ensured that there was not too much overlap between the neighboring basis functions, and by setting the off-diagonal entries of $\boldsymbol{\Sigma}$ to zero, the resulting RBFs had circular level sets.

After training, the control agent was able to reach the goal from arbitrary initial positions, usually taking at most 130 steps to reach the goal. We visualize the

[12] We also ran experiments using tile coding, another method for converting a continuous state space into a sparse feature vector amenable to linear TD methods, but the results were very similar to the NRBF case. With a sufficient number of tiles, the policies learned were essentially the same, and the agents had comparable performance. This similarity also held when learning the value via NRBFs and the variance using tile coding (or vice-versa), and likewise when using a different representation for the control policy and the value/variance estimators.

resulting behavior by plotting the agent's trajectories from various different starting states in Fig. 5.11.

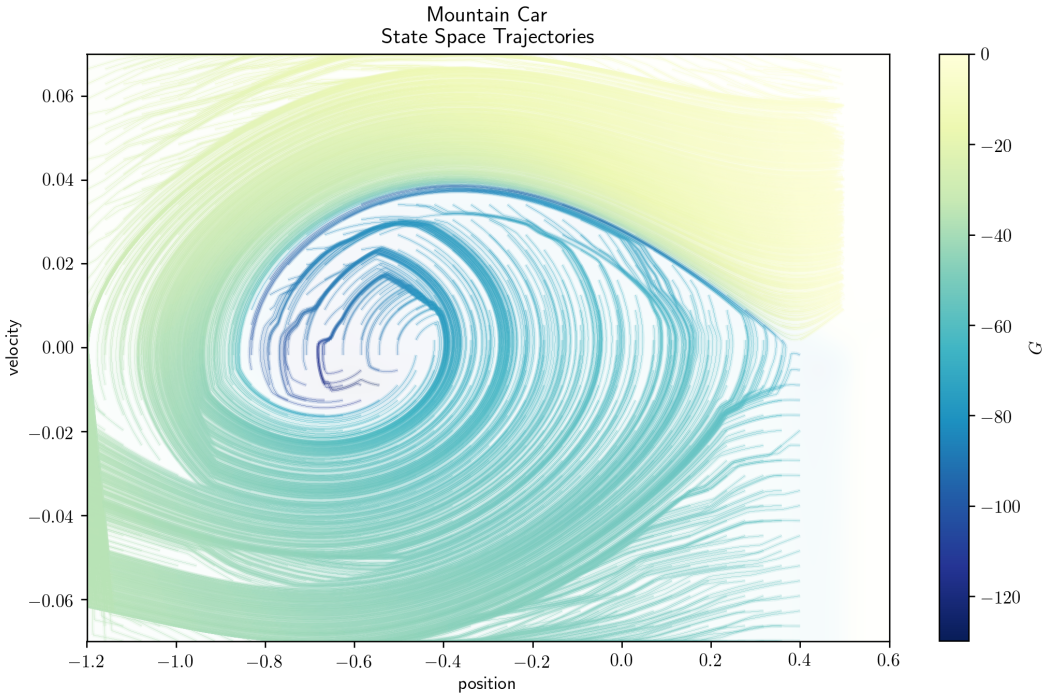

Figure 5.11: *Trajectories produced by the control policy from various initial starting positions Mountain Car. Each point in the run is colored according to the Monte Carlo return from that point, which is proportional to the number of steps before reaching the goal.*

We then froze the policy and used it to generate datasets for policy evaluation. Figure 5.11 indicates that the agent tends to proceed along certain flows through the state space, and some regions are only visited if the agent starts in them. This becomes a bit of a problem if the actions are selected greedily (*e.g.*, with $\epsilon = 0$), as most trajectories originating from a particular state will be identical[13]. As we are interested in how well the algorithms approximate variance, we opted to set $\epsilon = 0.2$, in order to inject more stochasticity into the policy and to induce the trajectories to cover more of the state space.

The standard Mountain Car setup initializes the agent randomly with position between -0.6 and -0.4 and zero velocity. In order to focus on the parts of the state space that are accessible with the standard initialization, the datasets we generated had initial states restricted to the sub-interval $[-1.0, 0.2] \times [-0.02, 0.04]$. We sampled from a 40×40 grid over that sub-interval, restarting from the same state ten times

[13]Even with just *mostly* greedy action selection, the trajectories tend to be fairly similar.

for a total of $16,000$ episodes, yielding about 1.5 million transitions in total.

The data thus generated was used to evaluate the algorithms with different hyperparameters. TD(λ) provided the value estimates used by both DVTD and VTD over different settings of λ, $\bar{\lambda}$, and κ. All algorithms shared the same stepsize of $\alpha = 0.05$.

This produced a fairly large quantity of data to analyze[14]. To avoid boring the reader, we omit presenting most of the results, merely stating that what *is* included is representative of the algorithms' behavior across the experiments we performed.

5.2.2 Evaluation Methods

For the Tamar Chain experiments (Section 5.1), we could calculate the true value function and variance analytically, which provided a reasonable benchmark for evaluating our agents' performance. Mountain Car has a continuous state space[15], which makes determining v_π and u_π substantially harder.

When the true value function is unavailable, we normally fall back on an approximation of v_π, or measure our performance a more readily available error signal, such as the *Mean Square Return Error*[16] The advantage of $\overline{\text{RE}}$ is that it doesn't require knowledge of v_π, allowing us to measure the performance of value-estimating algorithms using only empirical data. However, we are keen to gauge how well DVTD estimates the variance, which is regrettably defined in terms of v_π: $\text{Var}\,[G_t] = \mathbb{E}_\pi[(G_t - v_\pi(S_t))^2]$. Accordingly, we elect to make an approximation for v_π and use it to estimate the variance, using this approximation as the benchmark for DVTD (and VTD).

One common approximation method is *grid discretization*, which entails dividing the state space into a grid of n-dimensional cells. Each state is mapped to a cell,

[14]Furthermore, the experimental setup that we settled on was obtained through exploratory work where we iterated through many variations of parameter settings, features, and initial conditions. The total amount of Mountain Car data thus produced was on the order of terabytes, even under compression.

[15]At least in theory— the implementation we used represented position and velocity with 64-bit floats, meaning the state space was ultimately discrete. Even so, there are approximately 10^{87} distinct coordinate pairs $(s_x, s_v) \in \mathcal{S}$, which is still rather daunting.

[16]Recall (from (4.40)) that $\overline{\text{RE}}(\hat{v}) \stackrel{\text{def}}{=} \mathbb{E}_\pi[(G_t - \hat{v}(s))^2]$.

with nearby states tending to share the same cell. Then we can approximate state functions (*e.g.*, v_π) by taking an average of the sampled values of those functions with respect to each cell. With a sufficiently fine discretization, this tends to accurately capture the behavior of the function being approximated[17].

For our Mountain Car experiments, we chose to use a uniform discretization, decomposing each dimension of the state space into the same number of sub-intervals. The grid-based approximation of the value, second moment, and variance is then:

$$v_{\text{GRID}}(s) \stackrel{\text{def}}{=} \text{Average}[G_t | S_t \in \chi(s)], \tag{5.9a}$$

$$v^{(2)}_{\text{GRID}}(s) \stackrel{\text{def}}{=} \text{Average}[G_t^2 | S_t \in \chi(s)], \tag{5.9b}$$

$$u_{\text{GRID}}(s) \stackrel{\text{def}}{=} v^{(2)}_{\text{GRID}}(s) - \left(v_{\text{GRID}}(s)\right)^2, \tag{5.9c}$$

where $\chi(s)$ is an indexing function that maps states to their corresponding cell within the grid. We plot the value function approximation for uniform grid discretization with varying refinement in Fig. 5.12.

As can be seen in Fig. 5.12, choosing the grid's refinement is a bit of a balancing act: if it's too coarse we lose detail, but if it's too fine then many cells will not be visited. With a 64×64 grid we achieve a compromise between coverage of the state space and level of detail, so that discretization would seem to be a reasonable choice for approximating v_π.

Estimating the return's variance with grid discretization via (5.9c) produces similar results as when estimating its value, as can be seen in Fig. 5.13.

Most of the states evince relatively low variance, except along a spiral-shaped portion, where the variance is higher ($\approx 750-1250$). This spiral seems to correspond to a possible discontinuity in the value function estimates in Fig. 5.12, which would seem to be confirmed by the trajectory plot in Fig. 5.11. In states near the spiral, the agent may need to make an additional pass to build up more speed before it can reach the goal, but its representation is not up to the task of identifying whether this is the case or not. The effect is magnified from setting $\epsilon = 0.2$, as the agent

[17] With some caveats: the function in question should ideally be continuous within each cell, and each cell should contain enough samples for the average to be representative.

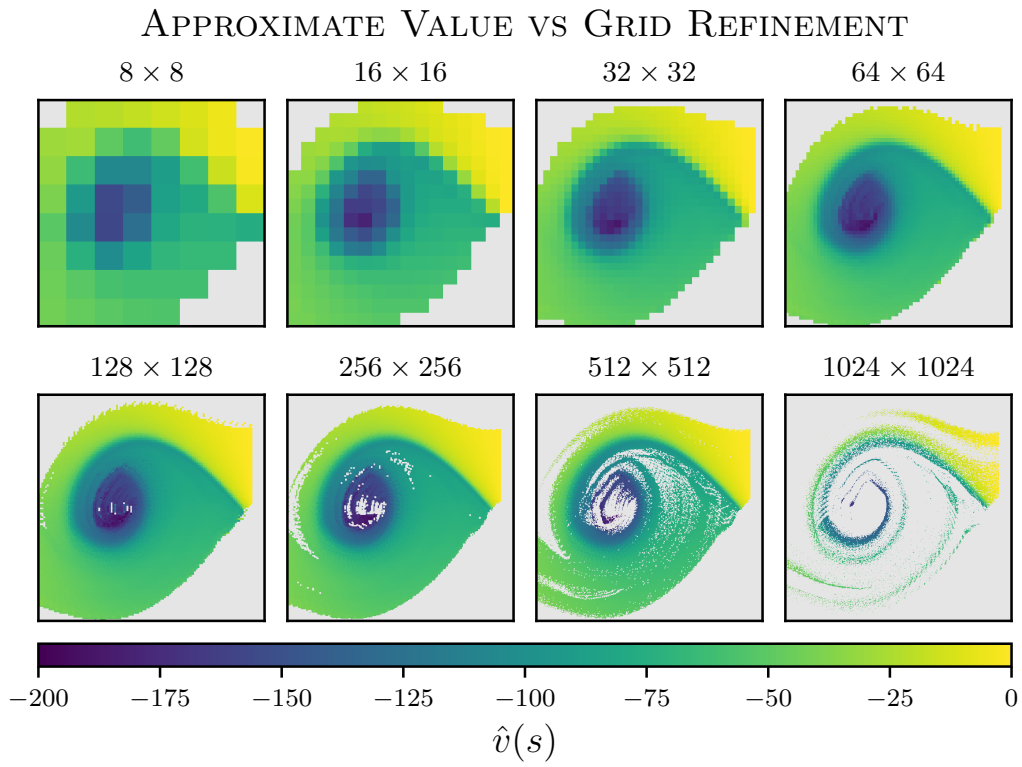

Figure 5.12: *The effects of using various levels of discretization fineness for value function approximation in Mountain Car. For each level of discretization, we calculate $\hat{v}$ following* (5.9a)*. A cell's colour reflects the average of the returns observed from that state; when no samples for a cell were available, we leave it uncoloured.*

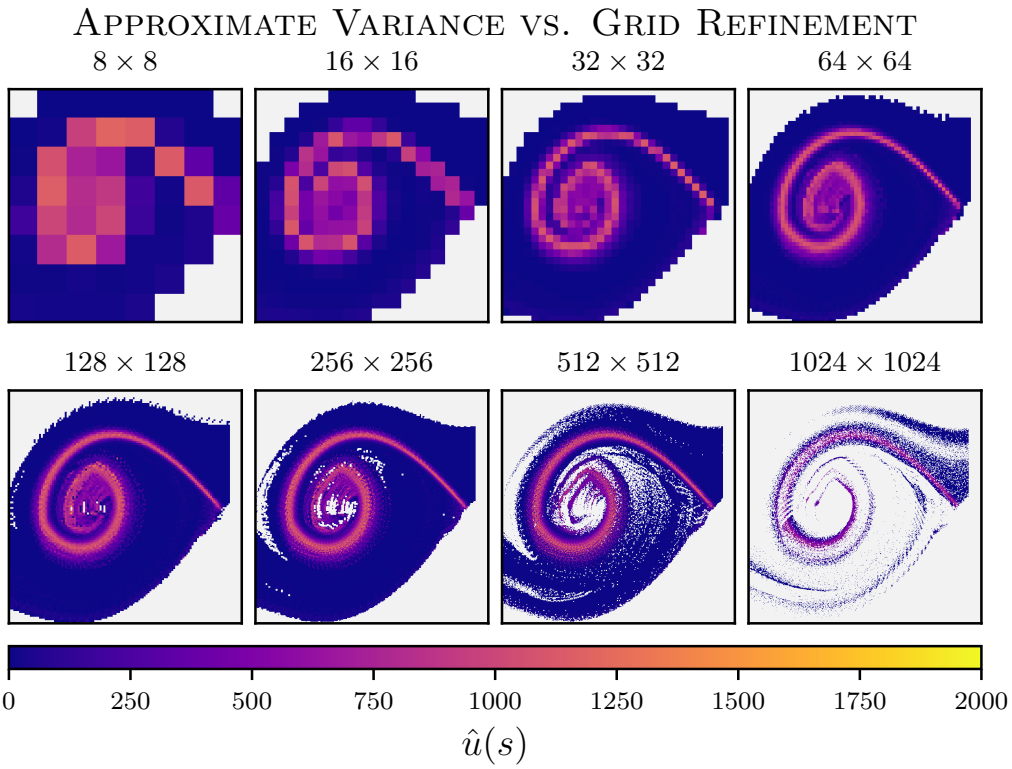

Figure 5.13: *Effects of discretization fineness on variance approximation in Mountain Car. For each level of discretization, we calculate* $\hat{u}$ *using* (5.9c)

may indeed be able to reach the goal without backtracking on if it acts greedily, but randomly selecting an inopportune action might force it to make another pass. This would tend to cause more variability in regions where (under the greedy policy) it will barely make it to the goal, while states where it has a comfortable margin for error will be less affected.

5.2.3 Performance

First, we quickly check that TD(λ) learns a reasonable estimate of the value function, before moving on to the variance estimators. We also compare the value function from TD(λ) with the least squares solution, as that is in some sense the best solution under the given features. The results are plotted in Fig. 5.14 for $\lambda = 0.9$.

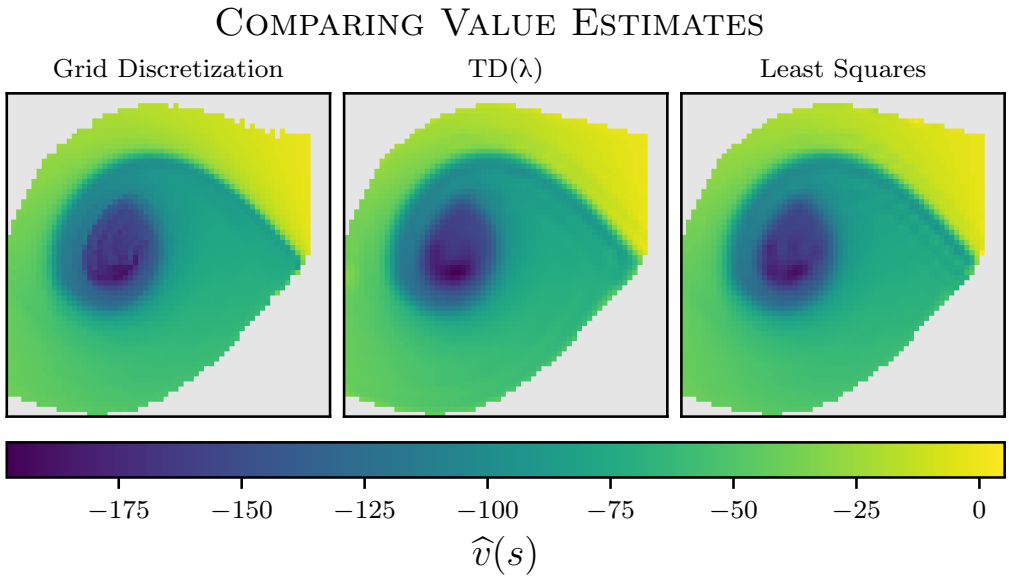

Figure 5.14: *A comparison of value function estimates for Mountain Car via three different approaches: a* 64×64 *grid discretization, TD(λ) with* $\lambda = 0.9$*, and the least squares solution.*

The value functions agree to the extent that they almost visually indistinguishable, which inspires confidence in our approximation for v_π as well as the performance of our TD(λ) implementation. For other values of λ, TD returned similar value functions, with performance tending to increase along with λ, as can be seen in Fig. 5.15.

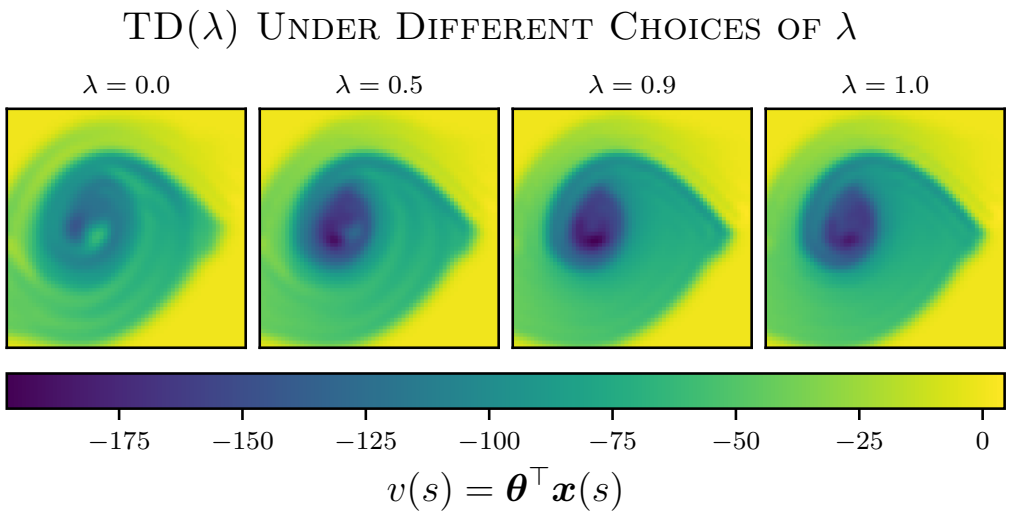

Figure 5.15: *The value functions learned by TD(λ) with different choices of λ. While generally similar, higher values of λ seem to produce a crisper separation between the different parts of the state space.*

For estimating variance, DVTD performed capably, although (as discussed earlier) it is difficult to make this statement precise without access to the true value function v_π. Instead we can compare it with the approximation found via discretization, as well as the least squares variance estimate. We plot the various approximations side-by-side in Fig. 5.16.

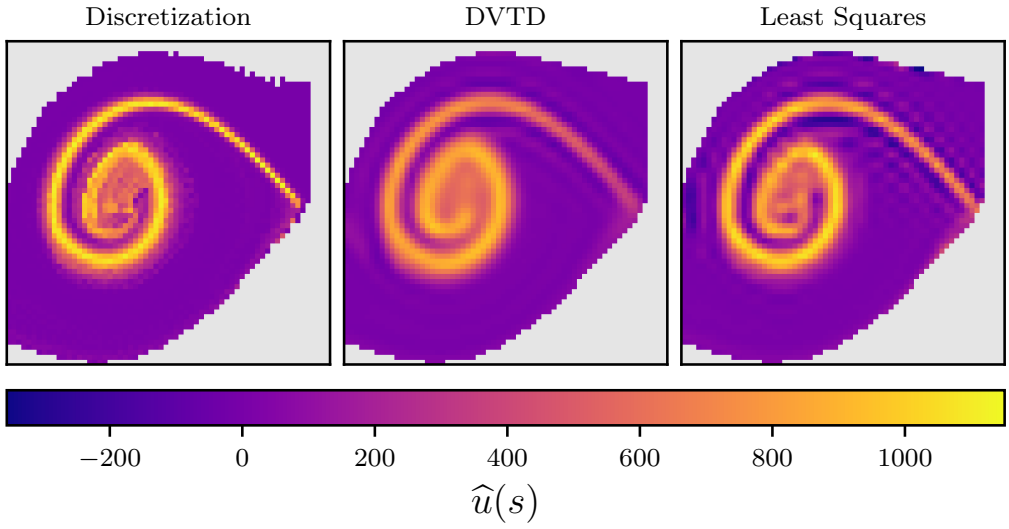

Figure 5.16: *We plot estimates for the variance of the return in Mountain Car using three different approaches. First, a 64×64 grid discretization, then the estimates from DVTD (with $\lambda = \bar{\lambda} = 0.9$), and finally the least squares approximation for variance.*

The results from DVTD show high agreement with those obtained from the other methods. Its estimates appear a bit blurrier than those from the grid discretization,

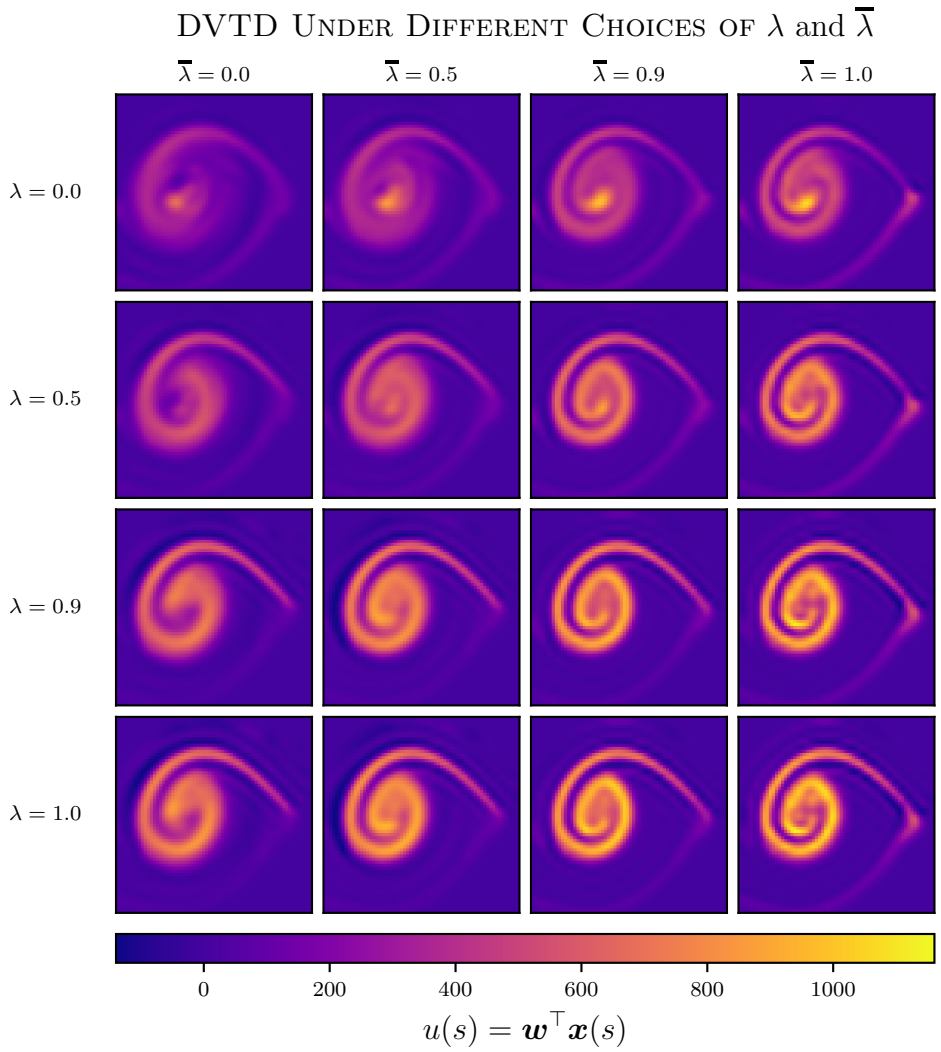

Figure 5.17: *Estimates for the variance of the return in Mountain Car, from DVTD with different choices for λ and $\bar{\lambda}$. With less bootstrapping (higher λ or $\bar{\lambda}$), the estimates become noticeably sharper.*

but that can be attributed to the fact that DVTD is using far fewer features[18].

However, as in our other experiments in Section 5.1, we found that the hyperparamters needed tuning to achieve the best result. In Fig. 5.17 we plot the approximations found by DVTD with different choices of λ and $\bar{\lambda}$.

It becomes apparent that DVTD benefits from having a more accurate value function (*i.e.*, higher λ), but even with $\lambda = 0$ it can still produce a credible estimate for the variance if $\bar{\lambda}$ is close to one. On the other hand, with $\lambda = 1$ it seems to work reasonably well for any choice of $\bar{\lambda}$. It produced the best estimates when both

[18]DVTD used a $30 \times 30 = 900$ NRBF grid, while the discretization had $64 \times 64 = 4096$ cells.

bootstrapping hyperparameters were closer to one, and was the least accurate when $\lambda = \bar{\lambda} = 0$. Notably, it did not tend to produce negative estimates except as a sort of ringing artifact near regions of high variance.

For comparison, our attempts to use VTD for variance approximation produced substantially worse results. We tried a variety of hyperparameter settings[19] and were unable to produce reasonable estimates. VTD's approximation for various λ and $\bar{\lambda}$ are plotted in Fig. 5.18.

No choice of bootstrapping lead to particularly good estimates, although the spiral pattern was always somewhat in evidence. For most choices of $\bar{\lambda}$ and λ, VTD's estimate was obviously incorrect for large portions of the state space. This was most noticeable for states corresponding to the mountain's "base", whose value tends to be the lowest (as it takes the most time to reach the goal from those states). This suggests that numerical instability was to blame, with relatively small errors in the value or second moment estimates leading to large errors when calculating variance[20]. Even in the best-case scenario, with $\lambda = \bar{\lambda} = 0.9$, VTD produced a somewhat reasonable result, although it has noticeable ringing artifacts.

5.2.4 Summary

DVTD once again manages to produce creditable estimates for the variance of the return. The estimates it produces are dependent on having a good value function, requiring a bit of care when selecting λ, although for $\lambda = 0$ it can still achieve a decent approximation so long as $\bar{\lambda}$ is relatively high. While some hyperparameter settings were better than others, in none of our experiments did we find DVTD to diverge or produce wildly incorrect estimates. In contrast, VTD tended to be much less stable, and produced worse estimates for the variance when compared to DVTD for the various values of λ and $\bar{\lambda}$ we tested.

[19] Including varying the stepsize, although we don't include those results here.

[20] For example, the states near the base of the mountain have values around -180, meaning that (180 ± 9) spans 29241 to 35721, a spread of about 6000. The highest estimate for variance across all states using any of the other methods we tried was around 1200, meaning that a $\pm 5\%$ difference can cause VTD's to over- or underestimate by five times the *range* of the other approaches.

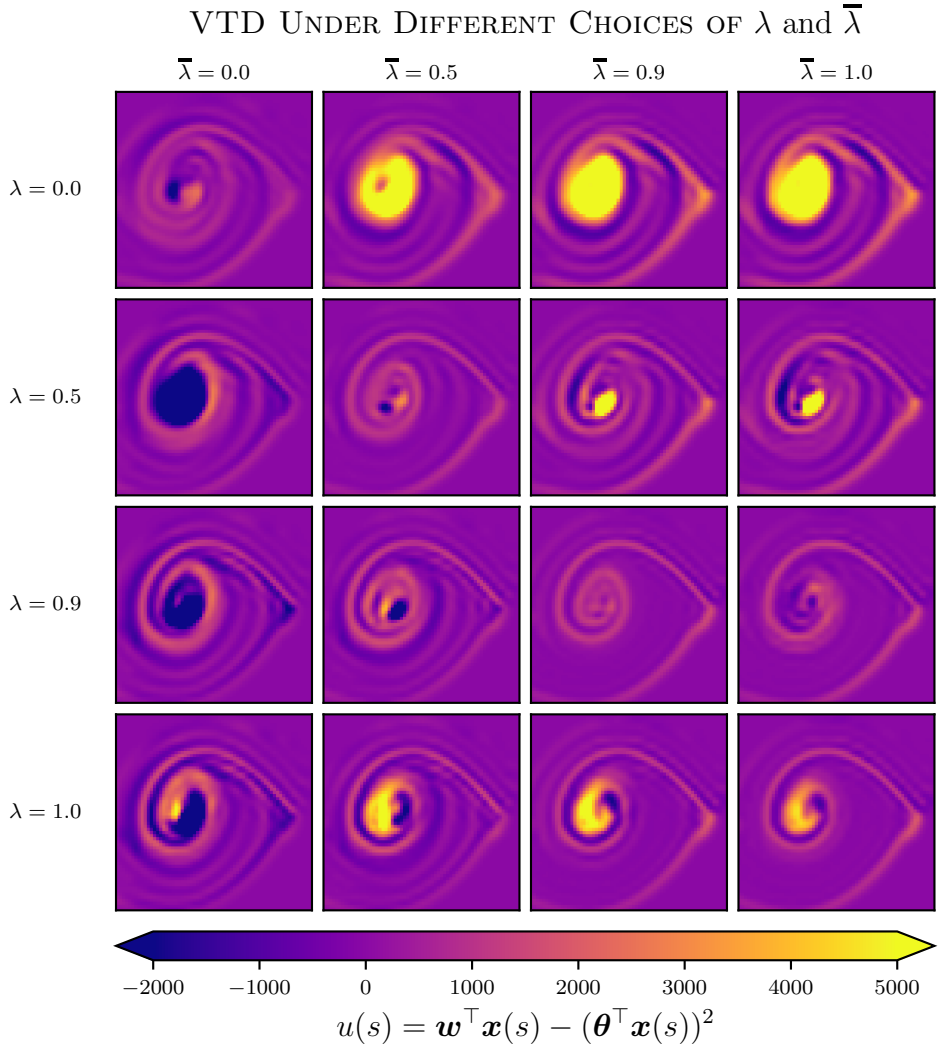

Figure 5.18: *Estimates for the return's variance in Mountain Car learned by VTD under different choices for* λ *and* $\bar{\lambda}$. ***For display purposes, we clipped the estimates to lie within*** $[-2000, 5000]$. *The actual range observed was between* -20000 *and* 25000.

Chapter 6

Conclusions & Future Work

We conclude this thesis with a summary of its contributions and a discussion of some avenues for future research.

6.1 Summary

This thesis was motivated by the desire to extend reinforcement learning algorithms to estimating quantities beyond the expected return while retaining the advantages of temporal difference methods.

In Chapter 3, we outlined why learning more elaborate predictions might be useful, with a particular focus on learning the moments of the return. We described how estimates of the return's moments could then be used to predict functions of the return using Taylor series or parametric methods. In the process, we generalized a TD-style algorithm for estimating the second moment of the return to arbitrary higher moments.

We then turned our attention towards the problem of estimating the return's second central moment, that is, the variance. A good estimate of the return's variance has numerous uses; however, the existing approaches came with certain drawbacks. The second moment method, while attractive from a theoretical standpoint, tends to suffer from numerical issues in practice. Algorithms that learn to approximate the return's distribution tend to require substantially more computational resources relative to vanilla TD(λ).

To avoid those disadvantages, we instead sought to estimate the variance *directly*. In Chapter 4, we described the δ^2-return, an approximation target formed from the

appropriately discounted sum of temporal difference errors. We showed that given an accurate value function, the δ^2-return corresponds to the variance of the return, and was in fact had the same expected value as the targets used by the second moment methods. Additional analysis explored the properties of this target when the value function is biased and how it can be modified to approximate the variance of arbitrary λ-returns, in addition to the usual Monte Carlo return. Some of the results in this chapter may be of independent interest for RL researchers.

We then described an algorithm (in Section 4.5) for learning the variance using the δ^2-return using temporal difference methods, called Direct Variance Temporal Difference Learning, or DVTD. Similar to TD(λ), DVTD is an efficient online learning algorithm, which allows the agent to learn an approximation of the return's variance directly and in an incremental fashion. Being based on TD(λ), it permits the use of eligibility traces, and can even be used to estimate the variance of different λ-returns than the one targeted by its value function approximator.

As DVTD is nonlinear, the usual contraction-based techniques for proving convergence did not apply directly. In response to this issue, we showed that a modified version of the algorithm that incorporates a projection step can be proved convergent using the ODE method. This projection step is generally unnecessary in practice, so the result indicates that the unmodified version will not diverge in typical use. We sketch the convergence proof in Section 4.6 and provide the complete result in Appendix A.

To validate our analysis and compare DVTD with alternatives, we then performed a number of experiments in some simple domains focusing on the linear function approximation setting. The results of our experiments are detailed in Chapter 5; however, the main findings can be summarized as follows:

1. Across many different hyperparameter settings, DVTD is converges more quickly relative to the equivalent second moment based algorithm VTD or the distributional RL algorithm C51.
2. When increasing the stepsize or bootstrapping, DVTD never diverged before VTD. As VTD is known to be convergent, this supports our contention that

DVTD is viable even without the projection step needed in the convergence proof.

3. C51 exhibited exemplary stability at the expense of a much slower convergence rate; however, when DVTD's stepsize was reduced to produce similarly stable estimates, DVTD still converged more quickly.
4. DVTD's asymptotes provided better estimates of the return's variance compared to the alternatives we tested, even under function approximation.
5. DVTD produced reasonable estimates for the return's variance across all hyperparameter settings. In contrast, VTD was very sensitive to the choice of bootstrapping. In the Mountain Car domain, VTD was prone to estimating implausibly high or even negative values for the variance of the return.

Further experiments in different domains or with different approximation architectures are needed to test whether DVTD's superior performance holds in general, but the available evidence suggests that it is the best option for estimating the return's variance if one wishes to use temporal difference methods.

6.2 Future Work

The obvious directions for future work involve applying the variance of the return (and possibly the return's higher moments) to improve the performance of our agents.

Modifying Behavior Using Variance Estimates After developing an algorithm for learning a value function, the next step is usually to apply it towards policy improvement in the control setting. Variance might be particularly useful here, as it can be used to enforce a notion of safety by avoiding actions with high variance, or conversely, to guide exploration by prioritizing high-variance policies. Some work in this vein (Jain, Khetarpal and Precup 2018) has already demonstrated how the squared TD error can be used as a proxy for the "controllability" of an agent, which the authors used to tune a policy gradient algorithm referred to as Safe Option-Critic.

They found incorporating a penalty from δ^2 could improve the agent's performance relative to the baseline in a number of domains within the Arcade Learning Environment. It seems likely that a similar algorithm making use of the full δ^2-return could further increase performance.

Higher Moment Estimation in Practice As discussed in Chapter 3, sometimes the "natural" formulation of the return does not capture salient aspects of the task. A better objective might be expressed as a function of the return, which could be approximated using estimates of the return's moments.

Before this approach becomes practical we need to better characterize algorithms for learning these higher moments; our experiments with VTD suggest that further work is needed to achieve adequate stability and accuracy. We would seek to identify situations where these algorithms will produce reliable estimates of the moments of the return and to determine tighter performance bounds.

However, there is some indication that these higher moments can in fact be useful. In the course of writing this thesis, we compared estimates of the return's distribution from C51 versus a normal approximation using DVTD's value and variance estimates. An example from the Tamar Chain environment is plotted in Fig. 6.1.

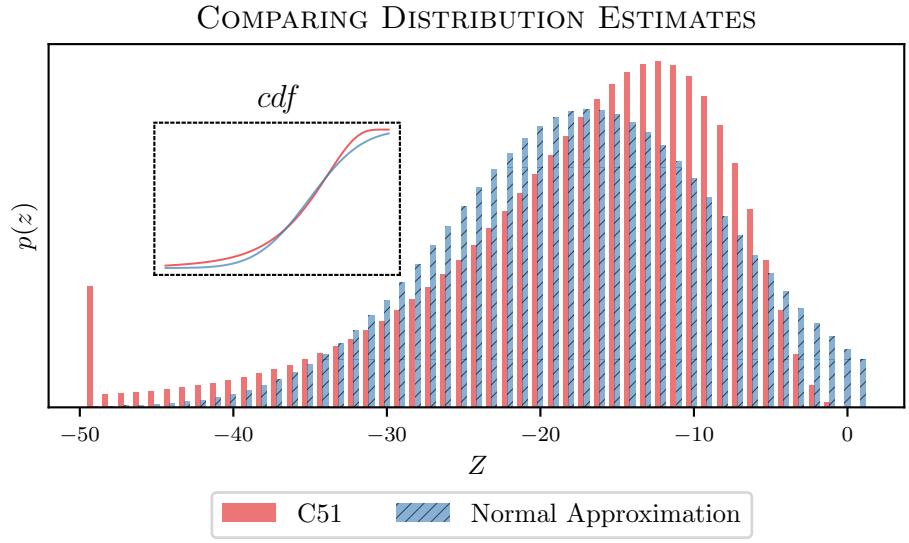

Figure 6.1: *Estimates for the distribution of the return from a state (specifically, State 2 in the Tamar Chain MDP) under linear function approximation. We used the value and variance estimates from DVTD to approximate the return's distribution as a standard normal, comparing it with the distribution estimate found by C51.*
The normal approximation is similar to C51's estimate, but does not capture the distribution's left-tailedness and has a visibly different mode.

In this case the parametric approximation was not particularly accurate, but this is to be expected given that C51 uses far more parameters and is specifically designed to approximate distributions.

However, upon incorporating *one* additional moment (the skew), we found that the parametric approximation improved significantly; a fitted skew normal largely agrees with the distribution estimated by C51, as can be seen in Fig. 6.2.

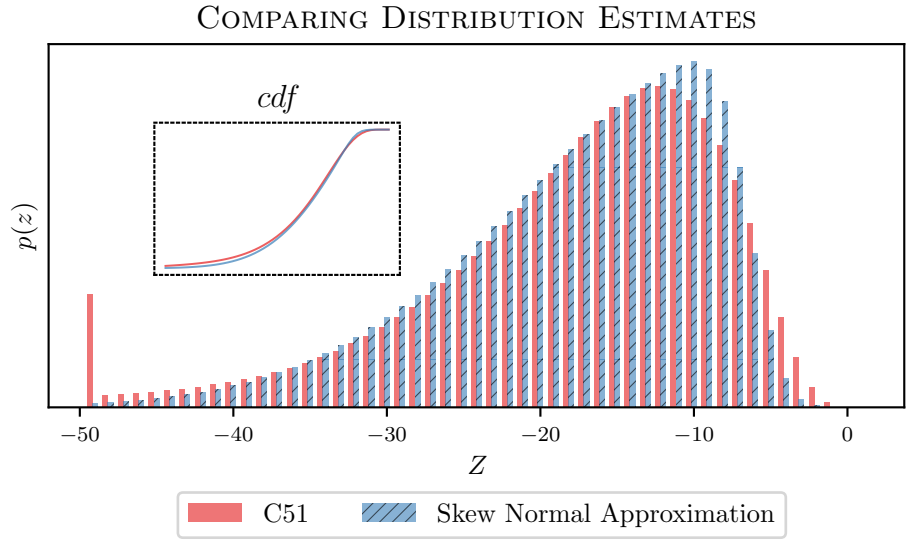

Figure 6.2: *The distribution of the return from a state (specifically, State 2 in the Tamar Chain MDP), as estimated by C51 (under linear function approximation) and a fitted skew-normal. Inset, we also provide the associated cumulative distribution functions.*
They show a surprising amount of agreement, with similar modes and appropriately decaying tails. The estimated distribution from C51 has some concentration of probability mass at the left edge of the support, indicating that a portion of the returns were more negative than could be represented.

This sort of accuracy might not be universally attainable (particularly if the return distribution is multimodal) but it suggests that moment-based methods could be competitive with the more elaborate distributional approach.

Alternatively, comparing the estimate for the variance from, say, DVTD with the variance estimated by C51 would allow us to check that they are performing properly. It might also be possible to accelerate distributional algorithms by initializing them estimated distributions obtained from the faster moment learning algorithms.

Appendix A

DVTD Convergence Details

Here we present a more detailed version of DVTD's convergence proof from Section 4.6. We use the two-timescale convergence results from Borkar 2008, described in Section 2.6, in particular Theorem 2.2. Essentially, we show that a version of DVTD[1] satisfies the conditions ($\mathcal{A}$. 2.1 to 2.9) of the general convergence theorem, implying Theorem 4.6.

For Theorem 4.6 to hold, we had to make some initial assumptions, in particular that the transitions were sampled i.i.d. ($\mathcal{A}$. 2.2) from the on-policy distribution and that the rewards and features were bounded[2] ($\mathcal{A}$. 4.1). Furthermore, we assume that the feature matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$ is full rank.

In terms of notation, we specify transitions as (s_n, r_n, s'_n), and we use $\mathbf{x}_n \stackrel{\text{def}}{=} \mathbf{x}(s_n)$, $\mathbf{x}'_n \stackrel{\text{def}}{=} \mathbf{x}(s'_n)$, and $\gamma_n \stackrel{\text{def}}{=} \gamma(s'_n)$. We use $\gamma = \max_{s \in \mathcal{S}} \gamma(s)$, to allow us to simultaneously establish results for both the GVF and constant discounting settings. Unless otherwise specified, we take $\|\cdot\|$ to be the Euclidean norm, with $\|\cdot\|_{\mathbf{M}}$ the *weighted* norm with respect to some matrix $\mathbf{M}$[3].

We also take for granted that the stepsize sequence is chosen appropriately ($\mathcal{A}$. 2.3), with the TD component corresponding to the slow timescale and identifying the DV component with the fast timescale. Additionally, the DVTD variant we analyze

[1] A summary of the various quantities involved is provided in Table A.1 (Page 109.

[2] With $\|\mathbf{x}(s)\| \leq K_b$ for all $s \in \mathcal{S}$ and $|r_n| < K_b$ for $n \geq 0$.

[3] *e.g.*, $\|\mathbf{z}\|_{\mathbf{M}} \stackrel{\text{def}}{=} (\mathbf{z}^\top \mathbf{M} \mathbf{z})^{1/2} = \|\mathbf{M}\mathbf{z}\|$.

incorporates a projection step:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\Upsilon}\Big(\boldsymbol{\theta}_n + \alpha_n[\mathbf{g}(\boldsymbol{\theta}_n) + \mathbf{M}_{n+1}]\Big), \tag{4.59}$$

$$\mathbf{w}_{n+1} = \boldsymbol{\Upsilon}\Big(\mathbf{w}_n + \beta_n[\mathbf{h}(\mathbf{w}_n) + \mathbf{N}_{n+1}]\Big), \tag{4.60}$$

where $\boldsymbol{\Upsilon} : \mathbb{R}^d \to C$ is a projection that restricts the iterates to a compact set $C \subset \mathbb{R}^d$ chosen such that it contains all possible equilbria for the associated ordinary differential equations:

$$\begin{aligned} \dot{\boldsymbol{\theta}} &= \mathbf{g}(\boldsymbol{\theta}) &&= -\mathbf{A}[\boldsymbol{\theta} - \boldsymbol{\theta}^*] &&= \mathbf{b} - \mathbf{A}\boldsymbol{\theta}, \\ \dot{\mathbf{w}} &= \mathbf{h}(\boldsymbol{\theta}, \mathbf{w}) &&= -\tilde{\mathbf{A}}[\mathbf{w} - \mathbf{w}^*(\boldsymbol{\theta})] &&= \tilde{\mathbf{b}}(\boldsymbol{\theta}) - \tilde{\mathbf{A}}\mathbf{w}. \end{aligned}$$

If a point is an equilibrium in the unprojected system, then it remains an equilibrium under projection. By employing a projection, we ensure that the update equations (and their corresponding ODEs) will be Lipschitz, which is immensely helpful for proving that (4.59) and (4.60) converge to a unique solution[4] We tacitly assume that the initial $\boldsymbol{\theta}_0$ and $\mathbf{w}_0$ are chosen to be within C in order to streamline the exposition.

With the preceding in mind, we can then establish the truth of the remaining assumptions.

Lemma A.1 (Bounded Iterates)

The iterates $\boldsymbol{\theta}_n$ and $\mathbf{w}_n$ are bounded in accordance with $\mathcal{A}$. 2.7:

$$\sup_n(\|\mathbf{w}_n\| + \|\boldsymbol{\theta}_n\|) \leq \infty \quad \text{a.s.} \tag{A.1}$$

Proof (Lemma A.1):

The bound is immediately evident due to the projection step. We defined $K_C \stackrel{\text{def}}{=} \sup_{\mathbf{z}\in C} \|\mathbf{z}\|$, and since the updates in (4.59) and (4.60) are contained within C for all $n \geq 0$, then we have

$$\|\boldsymbol{\theta}_n\|, \|\mathbf{w}_n\| \leq K_C < \infty \quad \forall\ n \geq 0 \tag{A.2}$$

■

[4]Further information on this strategy is available in Nagurney and Zhang 1996, Chapters 2-3; examples of applications in a reinforcement learning context can be found in Yu 2017; Sutton, Maei et al. 2009.

For later convenience, we establish the positive definiteness of $\mathbf{A}$ and $\tilde{\mathbf{A}}$ as the consequence of a more general result:

Lemma A.2 (Positive Definiteness of Update Matrices)

For $k \in \mathbb{N}$ such that $k > 0$, let $\mathbf{A}_{(k)} \in \mathbb{R}^{d \times d}$ be defined as:

$$\mathbf{A}_{(k)} \stackrel{\text{def}}{=} \mathbf{X}^\top \mathbf{D}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^{(k)})\mathbf{X}, \tag{A.3}$$

where $\mathbf{P} \in \mathbb{R}N \times N$ is an irreducible stochastic matrix, with stationary distribution $\mathbf{d}$, $\mathbf{D} = \mathsf{diag}(\mathbf{d})$, and $\boldsymbol{\Gamma} \in \mathbb{R}N \times N$ is a diagonal matrix with $0 \leq \Gamma_{ii} \leq 1$ which has at least one entry less than one.

If $\mathbf{X} \in \mathbb{R}^{N \times d}$ is full rank, then $\mathbf{A}_{(k)}$ is positive definite.

Proof (Lemma A.2):

From our assumption that $\mathbf{X}$ is full rank, we have for all $\mathbf{y} \in \mathbb{R}^d$ that $\mathbf{z} = \mathbf{X}\mathbf{y} \neq \mathbf{0}$ unless $\mathbf{y} = \mathbf{0}$. We can then observe that for any positive definite matrix $\mathbf{M}$: $\mathbf{z}^\top \mathbf{M}\mathbf{z} = \mathbf{y}^\top \mathbf{X}^\top \mathbf{M}\mathbf{X}\mathbf{y} > 0$ for $\mathbf{y} \neq \mathbf{0}$.

Therefore we just need to show that $\mathbf{D}(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma}^{(n)})$ is positive definite. This can be seen as a consequence of diagonal dominance (as in Sutton, Mahmood and White 2015, pg. 6), stemming from the fact that $\mathbf{P}$ is a stochastic matrix, and for $n > 0$, the matrix $\mathbf{P}\boldsymbol{\Gamma}^{(n)}$ is substochastic.

Thus $\mathbf{A}_{(k)}$ is positive definite for $k \in \mathbb{N}$, $k > 0$. ■

Note that $\mathbf{A}_{(1)} = \mathbf{A}$ and $\mathbf{A}_{(2)} = \tilde{\mathbf{A}}$ are therefore both positive definite.

We will also make use of bounds on the TD error:

Lemma A.3 (Bounding $\delta(\boldsymbol{\theta})$)

Assume $\mathcal{A}$. 2.2 and 4.1 hold and that the TD component updates as in (4.59). Let $K_C \stackrel{\text{def}}{=} \sup_{\mathbf{z} \in C} \|\mathbf{z}\|$. Then:

$$|\delta_n(\boldsymbol{\theta})| = \left|r_n + (\gamma_n \mathbf{x}'_n - \mathbf{x}_n)^\top \boldsymbol{\theta}\right| \leq K_b(1 + (1+\gamma)\|\boldsymbol{\theta}\|) \leq K_\delta, \tag{A.4}$$

where we define $K_\delta \stackrel{\text{def}}{=} K_b + (1+\gamma)K_b K_C$

In addition, for the expected TD error vector, $\boldsymbol{\delta}(\boldsymbol{\theta})$, we have:

$$\|\boldsymbol{\delta}(\boldsymbol{\theta})\|_{\mathbf{D}} \leq K_b(1 + (1+\gamma)\|\boldsymbol{\theta}\|_{\mathbf{D}}) \leq K_\delta. \tag{A.5}$$

Proof (Lemma A.3):

The proof is simple given our assumptions. Starting from the definition,

$$\begin{aligned} |\delta_n(\boldsymbol{\theta})| &= \left|r_n + (\gamma_n \mathbf{x}'_n - \mathbf{x}_n)^\top \boldsymbol{\theta}\right| \\ &\leq |r_n| + \left|\gamma_n (\mathbf{x}'_n)^\top \boldsymbol{\theta}\right| + \left|\mathbf{x}_n^\top \boldsymbol{\theta}\right| \\ &\leq K_b + \gamma \|\mathbf{x}'_n\| \|\boldsymbol{\theta}\| + \|\mathbf{x}_n\| \|\boldsymbol{\theta}\| \\ &\leq K_b(1 + (1+\gamma)\|\boldsymbol{\theta}\|), \end{aligned} \tag{A.6}$$

from the bounds on the features and the rewards. Noting that $\boldsymbol{\theta}_n \in C$ for all $n \geq 0$, we have $K_C \geq \|\boldsymbol{\theta}\|$, and therefore:

$$|\delta_n| \leq K_b(1 + (1+\gamma)\|\boldsymbol{\theta}\|) \leq K_b(1 + (1+\gamma)K_C) = K_\delta \tag{A.7}$$

as claimed.

We can follow a similar procedure for the second part of the lemma:

$$\|\boldsymbol{\delta}(\boldsymbol{\theta})\|_{\mathbf{D}} = \|\mathbf{r} + (\mathbf{P}\boldsymbol{\Gamma} - \mathbf{I})\mathbf{X}\boldsymbol{\theta}\|_{\mathbf{D}} \leq \|\mathbf{r}\|_{\mathbf{D}} + \|(\mathbf{P}\boldsymbol{\Gamma} - \mathbf{I})\mathbf{X}\boldsymbol{\theta}\|_{\mathbf{D}}. \tag{A.8}$$

Observe that $\|\mathbf{r}\| \leq K_b$; for the other term we have:

$$\begin{aligned} \|(\mathbf{P}\boldsymbol{\Gamma} - \mathbf{I})\mathbf{X}\boldsymbol{\theta}\|_{\mathbf{D}} &\leq \|(\mathbf{P}\boldsymbol{\Gamma} - \mathbf{I})\mathbf{X}\|\boldsymbol{\theta}\|_{\mathbf{D}} \leq (\|\mathbf{P}\boldsymbol{\Gamma}\mathbf{X}\| + \|\mathbf{X}\|)\|\boldsymbol{\theta}\|_{\mathbf{D}} \\ &\leq (1+\gamma)K_b\|\boldsymbol{\theta}\|_{\mathbf{D}} \leq (1+\gamma)K_bK_C. \end{aligned} \tag{A.9}$$

Combining, we get the desired result:

$$\|\boldsymbol{\delta}(\boldsymbol{\theta})\|_{\mathbf{D}} \leq K_b + (1+\gamma)K_bK_C = K_\delta. \tag{A.10}$$

■

Lemma A.4 (Slow Component Convergence)

The slow component $\dot{\boldsymbol{\theta}} = \mathbf{g}(\boldsymbol{\theta}) = \mathbf{b} - \mathbf{A}\boldsymbol{\theta}$ has a globally asymptotically stable equilibrium $\boldsymbol{\theta}^*$, as required by $\mathcal{A}$. 2.9.

PROOF (LEMMA A.4):

From Lemma A.2 we have that $\mathbf{A}$ is positive definite. This implies that it is invertible, and therefore that $\boldsymbol{\theta}^* = \mathbf{A}^{-1}\mathbf{b}$ exists; furthermore we can write the ODE as $\dot{\boldsymbol{\theta}} = \mathbf{b} - \mathbf{A}\boldsymbol{\theta} = -\mathbf{A}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)$. By assumption, $\boldsymbol{\theta}^* \in C$, so now we just need to show that it is a globally asymptotically stable equilibrium of (2.34).

This can be demonstrated in a number of ways, but perhaps the most immediate is the Lipaunov method. Using $V(\boldsymbol{\theta}) = \|\boldsymbol{\theta} - \boldsymbol{\theta}^*\|^2$ as our Liapunov function, we have:

$$\frac{\mathrm{d}V}{\mathrm{d}t} = \langle \nabla_{\boldsymbol{\theta}} V, \dot{\boldsymbol{\theta}} \rangle = \sum_{i=1}^{d} \frac{\partial V}{\partial \theta_i} \partial g_i(\boldsymbol{\theta}) = -2(\boldsymbol{\theta} - \boldsymbol{\theta}^*)^\top \mathbf{A}(\boldsymbol{\theta} - \boldsymbol{\theta}^*), \tag{A.11}$$

and since $\mathbf{A}$ is positive definite, we have that $\dot{V}(\boldsymbol{\theta}) < 0$ for $\boldsymbol{\theta} \neq \boldsymbol{\theta}^*$ Clearly, $V(\boldsymbol{\theta}) > 0$ for $\boldsymbol{\theta} \neq \boldsymbol{\theta}^*$, and $V(\boldsymbol{\theta}^*) = \dot{V}(\boldsymbol{\theta}^*) = 0$, so $V(\cdot)$ is a valid Liapunov function for the slow component $\dot{\boldsymbol{\theta}}$.

Then the equilibrium $\boldsymbol{\theta}^*$ exists and is globally asymptotically stable. ■

Lemma A.5 (Lipschitz Functions)

The functions $\boldsymbol{\delta}^{(2)} : \mathbb{R}^d \to \mathbb{R}^N$, $\tilde{\mathbf{b}} : \mathbb{R}^d \to \mathbb{R}^d$, and $\mathbf{w}^* : \mathbb{R}^d \to \mathbb{R}^d$ defined via:

$$\begin{aligned} \boldsymbol{\delta}^{(2)}(\boldsymbol{\theta}) &= \mathop{\mathbb{E}}_{\pi}[\delta_n^2(\boldsymbol{\theta})], \\ \tilde{\mathbf{b}}(\boldsymbol{\theta}) &= \mathop{\mathbb{E}}_{\pi}[\mathbf{x}_n \delta_n^2(\boldsymbol{\theta})] = \mathbf{X}^\top \mathbf{D} \boldsymbol{\delta}^{(2)}(\boldsymbol{\theta}), \\ \mathbf{w}^*(\boldsymbol{\theta}) &= \tilde{\mathbf{A}}^{-1} \tilde{\mathbf{b}}(\boldsymbol{\theta}), \end{aligned} \tag{A.12}$$

are Lipschitz on C.

PROOF (LEMMA A.5):

From the earlier Lemma A.3, we have that $\boldsymbol{\delta} : \mathbb{R}^d \to \mathbb{R}^N$ is bounded on C with $\|\delta_n(\boldsymbol{\theta})\| \leq K_\delta$ for some $K_\delta \geq 0$. The function $\boldsymbol{\delta}(\cdot)$ is actually itself Lipschitz, since:

$$\begin{aligned} \|\boldsymbol{\delta}(\boldsymbol{\theta}_1) - \boldsymbol{\delta}(\boldsymbol{\theta}_2)\|_{\mathbf{D}} &= \|\mathbf{r} + (\mathbf{P}\boldsymbol{\Gamma} - \mathbf{I})\mathbf{X}\boldsymbol{\theta}_1 - \mathbf{r} - (\mathbf{P}\boldsymbol{\Gamma} - \mathbf{I})\mathbf{X}\boldsymbol{\theta}_2\|_{\mathbf{D}} \\ &= \|(\mathbf{P}\boldsymbol{\Gamma} - \mathbf{I})\mathbf{X}(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2)\|_{\mathbf{D}} \\ &\leq \|(\mathbf{I} - \mathbf{P}\boldsymbol{\Gamma})\mathbf{X}\|_{\mathbf{D}} \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_{\mathbf{D}} \\ &\leq (1 + \gamma) K_b \overset{105}{=} K_\delta. \end{aligned} \tag{A.13}$$

This can be used to demonstrate that $\boldsymbol{\delta}^{(2)}$ is Lipschitz on C.

For $\boldsymbol{\delta}_n^{(2)} : \mathbb{R}^d \to \mathbb{R}^d$, we have:

$$\delta_n^{(2)}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\pi}[\delta_n^2(\boldsymbol{\theta})] = \mathop{\mathbb{V}}_{\pi}[\delta_n(\boldsymbol{\theta})] + \mathop{\mathbb{E}}_{\pi}[\delta_n(\boldsymbol{\theta})]^2 = \nu^2 + \boldsymbol{\delta}^2(\boldsymbol{\theta}) = \nu^2 + \boldsymbol{\delta}(\boldsymbol{\theta}) \circ \boldsymbol{\delta}(\boldsymbol{\theta}), \tag{A.14}$$

where we use $\nu^2 = \mathbb{V}_\pi[\delta_n(\boldsymbol{\theta})]$, noting that it is constant given $\boldsymbol{\mathcal{A}}$. 2.2. Therefore:

$$\begin{aligned} \left\|\boldsymbol{\delta}^{(2)}(\boldsymbol{\theta}_1) - \boldsymbol{\delta}^{(2)}(\boldsymbol{\theta}_2)\right\|_{\mathbf{D}} &= \left\|\boldsymbol{\delta}^2(\boldsymbol{\theta}_1) - \boldsymbol{\delta}^2(\boldsymbol{\theta}_2)\right\|_{\mathbf{D}} \\ &= \left\|(\boldsymbol{\delta}(\boldsymbol{\theta}_1) + \boldsymbol{\delta}(\boldsymbol{\theta}_2)) \circ (\boldsymbol{\delta}(\boldsymbol{\theta}_1) - \boldsymbol{\delta}(\boldsymbol{\theta}_2))\right\|_{\mathbf{D}} \\ &\leq \|\boldsymbol{\delta}(\boldsymbol{\theta}_1) + \boldsymbol{\delta}(\boldsymbol{\theta}_2)\| \|\boldsymbol{\delta}(\boldsymbol{\theta}_1) - \boldsymbol{\delta}(\boldsymbol{\theta}_2)\|_{\mathbf{D}} \\ &\leq 2K_\delta \|\boldsymbol{\delta}(\boldsymbol{\theta}_1) - \boldsymbol{\delta}(\boldsymbol{\theta}_2)\|_{\mathbf{D}}, \end{aligned} \tag{A.15}$$

where we use the fact that $\boldsymbol{\delta}$ is bounded on C. Then we can see:

$$\left\|\boldsymbol{\delta}^{(2)}(\boldsymbol{\theta}_1) - \boldsymbol{\delta}^{(2)}(\boldsymbol{\theta}_2)\right\|_{\mathbf{D}} \leq 2K_\delta \|\boldsymbol{\delta}(\boldsymbol{\theta}_1) - \boldsymbol{\delta}(\boldsymbol{\theta}_2)\|_{\mathbf{D}} \leq 2K_\delta^2 \|\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2\|_{\mathbf{D}}. \tag{A.16}$$

So $\boldsymbol{\delta}^{(2)}(\cdot)$ is Lipschitz on C.

This immediately implies that $\tilde{\mathbf{b}}(\cdot)$ and $\mathbf{w}^*(\cdot)$ are also Lipschitz on C, since they are both linear maps with respect to $\boldsymbol{\delta}^{(2)}$. ■

Lemma A.6 (Fast Component Convergence)

The fast component $\dot{\mathbf{w}} = \mathbf{h}(\mathbf{w}(t), \boldsymbol{\theta})$ has a globally asymptotically stable equilibrium that is a function of $\boldsymbol{\theta}$, denoted $\mathbf{w}^*(\boldsymbol{\theta})$, where $\mathbf{w}^* : \mathbb{R}^d \to \mathbb{R}^d$ is Lipschitz continuous.

Proof (Lemma A.6):

We know from Lemma A.5 that $\mathbf{w}^*(\cdot)$ is Lipschitz on C, which suffices for our purposes thanks to the projection step.

From Lemma A.2, we have that $\tilde{\mathbf{A}}$ is positive definite and therefore invertible, so $\mathbf{w}^*(\boldsymbol{\theta}) = \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{b}}(\boldsymbol{\theta})$ is unique. By inspection, we can see that $\dot{\mathbf{w}} = \mathbf{h}(\boldsymbol{\theta}, \mathbf{w}) = \tilde{\mathbf{b}} - \tilde{\mathbf{A}}\mathbf{w} = 0$ when $\mathbf{w} = \mathbf{w}^*$. What remains is to show that this equilibrium is globally asymptotically stable.

As it turns out, we can use the same reasoning as in Lemma A.4. Let

$V_{\boldsymbol{\theta}}(\mathbf{w}) = \|\mathbf{w} - \mathbf{w}^*(\boldsymbol{\theta})\|^2$ be our Liapunov function for some (fixed) choice of $\boldsymbol{\theta}$. It is obviously positive for $\mathbf{w} \neq \mathbf{w}^*(\boldsymbol{\theta})$, and $V_{\boldsymbol{\theta}}(\mathbf{w}^*(\boldsymbol{\theta})) = \dot{V}_{\boldsymbol{\theta}}(\mathbf{w}^*(\boldsymbol{\theta})) = 0$. Examining the derivative with respect to time, we see:

$$\frac{\mathrm{d}V_{\boldsymbol{\theta}}}{\mathrm{d}t} = \langle \nabla_{\mathbf{w}} V_{\boldsymbol{\theta}}, \mathbf{h}(\boldsymbol{\theta}, \mathbf{w}) \rangle = -2(\mathbf{w} - \mathbf{w}^*(\boldsymbol{\theta}))^\top \tilde{\mathbf{A}}(\mathbf{w} - \mathbf{w}^*(\boldsymbol{\theta})) \leq 0. \tag{A.17}$$

It is in fact negative for $\mathbf{w} \neq \mathbf{w}^*(\boldsymbol{\theta})$ due to the aforementioned positive-definiteness of $\tilde{\mathbf{A}}$.

Therefore $\mathbf{w}^*(\boldsymbol{\theta})$ is a unique globally asymptotically stable equilibrium for $\dot{\mathbf{w}} = \mathbf{h}(\mathbf{w}(t), \boldsymbol{\theta})$ as claimed. ■

Lemma A.7 (Noise Bound)

The noise sequences $\{\mathbf{M}_n\}$ and $\{\mathbf{N}_n\}$ are bounded by some $K_m > 0$ such that:

$$\begin{aligned} \mathbb{E}[\|\mathbf{N}_{n+1}\|^2 | \mathcal{F}_n] &\leq K_m[1 + \|\mathbf{w}_n\|^2 + \|\boldsymbol{\theta}_n\|^2], \\ \mathbb{E}[\|\mathbf{M}_{n+1}\|^2 | \mathcal{F}_n] &\leq K_m[1 + \|\mathbf{w}_n\|^2 + \|\boldsymbol{\theta}_n\|^2]. \end{aligned} \tag{A.18}$$

Proof (Lemma A.7):

We first establish bounds on the TD component's noise.

$$\begin{aligned} \|\mathbf{M}_{n+1}\| &= \|\mathbf{x}_n \delta_n(\boldsymbol{\theta}_n) - (\mathbf{b} - \mathbf{A}\boldsymbol{\theta}_n)\| \\ &\leq \left\|\mathbf{x}_n(r_n + (\gamma_n \mathbf{x}'_n - \mathbf{x})^\top \boldsymbol{\theta}\right\| + \|\mathbf{b} - \mathbf{A}\boldsymbol{\theta}_n\| \\ &\leq \|\mathbf{x}_n\|(|r_n| + \gamma\|\mathbf{x}'_n\| + \|\mathbf{x}_n\|)\|\boldsymbol{\theta}_n\| + \|\mathbf{b}\| + \|\mathbf{A}\|\|\boldsymbol{\theta}_n\| \\ &\leq K_b^2(1 + (1+\gamma)\|\boldsymbol{\theta}\|) + \|\mathbf{b}\| + \|\mathbf{A}\|\|\boldsymbol{\theta}_n\| \\ &= K_b^2 + \|\mathbf{b}\| + (K_b^2(1+\gamma) + \|\mathbf{A}\|)\|\boldsymbol{\theta}\|, \end{aligned} \tag{A.19}$$

where we used $\boldsymbol{\mathcal{A}}$. 4.1, which also guarantees that $\|\mathbf{b}\|$ and $\|\mathbf{A}\|$ are finite. Let $\overline{K}_m = \max K_b^2 + \|\mathbf{b}\|, (K_b^2(1+\gamma) + \|\mathbf{A}\|)$. Then:

$$\|\mathbf{M}_{n+1}\| \leq \overline{K}_m(1 + \|\boldsymbol{\theta}\|). \tag{A.20}$$

From Young's inequality, $ab \leq a^p/p + b^q/q$ for $a, b \geq 0$ and $p, q > 1$ such that $1/p + 1/q = 1$. In particular, if $p = q = 2$, and $a = 1$, we have $b \leq \frac{1+b^2}{2}$. We can

therefore see that:

$$\|\mathbf{M}_{n+1}\|^2 \leq \overline{K}_m^2(1+\|\boldsymbol{\theta}\|)^2 = \overline{K}_m^2(1+\|\boldsymbol{\theta}\|^2+2\|\boldsymbol{\theta}\|) \leq 2\overline{K}_m^2(1+\|\boldsymbol{\theta}\|^2), \tag{A.21}$$

■

which gives us a bound on $\|\mathbf{M}_{n+1}\|$ similar to the lemma's statement. Setting $K_m = 2\overline{K}_m^2$ completes the proof.

A bound for $\|\mathbf{N}_{n+1}\|$ in terms of $\|\mathbf{w}\|$ can be derived in a similar manner by following essentially the same steps and making use of Lemma A.3.

Lemma A.8 (Martingale Noise)

The noise sequences $\{\mathbf{M}_{n+1}\}$ and $\{\mathbf{N}_{n+1}\}$ are martingale difference sequences.

Proof (Lemma A.8):

We first show that $\{\mathbf{M}_{n+1}\}$ and $\{\mathbf{N}_{n+1}\}$ have expectation zero, which follows from their definition and the i.i.d. assumption ($\mathcal{A}$. 2.2). We have:

$$\mathbf{M}_{n+1} = \mathbf{x}_n(r_n + (\gamma_n\mathbf{x}'_n - \mathbf{x})^\top\boldsymbol{\theta}_n) - \mathbf{b} - \mathbf{A}\boldsymbol{\theta}_n = \mathbf{x}_n\delta_n(\boldsymbol{\theta}_n) - \mathbf{g}(\boldsymbol{\theta}_n). \tag{A.22}$$

Examining the expectation of $\delta_n(\boldsymbol{\theta}_n)\mathbf{x}_n$ yields:

$$\begin{aligned}
\mathop{\mathbb{E}}_{\pi}[\mathbf{x}_n\delta_n] &= \mathop{\mathbb{E}}_{\pi}[\mathbf{x}_n(r_n + (\gamma_n\mathbf{x}'_n - \mathbf{x})^\top\boldsymbol{\theta}_n)] \\
&= \mathop{\mathbb{E}}_{\pi}[\mathbf{x}_n r_n] + \mathop{\mathbb{E}}_{\pi}[\mathbf{x}n(\gamma_n\mathbf{x}'_n - \mathbf{x})^\top\boldsymbol{\theta}_n)] \\
&= \mathbf{D}\mathbf{X}^\top\mathbf{r} + \mathbf{D}\mathbf{X}^\top(\boldsymbol{\Gamma}\mathbf{P}\mathbf{X} - \mathbf{X})\boldsymbol{\theta}_n \\
&= \mathbf{b} - \mathbf{A}\boldsymbol{\theta}_n,
\end{aligned} \tag{A.23}$$

and therefore:

$$\mathop{\mathbb{E}}_{\pi}[\mathbf{M}_{n+1}] = \mathop{\mathbb{E}}_{\pi}[\mathbf{x}_n\delta_n] - (\mathbf{b} - \mathbf{A}\boldsymbol{\theta}_n) = \mathbf{b} - \mathbf{A}\boldsymbol{\theta}_n - (\mathbf{b} - \mathbf{A}\boldsymbol{\theta}_n). \tag{A.24}$$

For $\mathbf{N}_{n+1}$ the proof is basically identical and therefore omitted.

We can further note that Lemma A.7 implies that their expectations are bounded with proportional to the iterates. Given that the iterates are *themselves* bounded to be within C, we have that $\{\mathbf{M}_{n+1}\}$ and $\{\mathbf{N}_{n+1}\}$ are bounded, and therefore square-integrable.

Therefore $\{\mathbf{M}_{n+1}\}$ and $\{\mathbf{N}_{n+1}\}$ are martingale difference sequences. ■

The combination of the preceding results implies that the iterates from (4.59) and (4.60) satisfy the conditions from Theorem 2.2 and therefore $(\boldsymbol{\theta}_n, \mathbf{w}_n) \to (\boldsymbol{\theta}^*, \mathbf{w}^*(\boldsymbol{\theta}^*))$ as $n \to \infty$.

DVTD ODE SUMMARY

TD(0) update	$\boldsymbol{\theta}_{n+1} = \boldsymbol{\Upsilon}\Big(\boldsymbol{\theta}_n + \alpha_n[r_n + \gamma\boldsymbol{\theta}_n^\top\mathbf{x}'_n - \boldsymbol{\theta}_n^\top\mathbf{x}_n]\mathbf{x}_n\Big)$	
	$= \boldsymbol{\Upsilon}\Big(\boldsymbol{\theta}_n + \alpha_n[\mathbf{g}(\boldsymbol{\theta}_n) + \mathbf{M}_{n+1}]\Big)$	(4.59)
DV update	$\mathbf{w}_{n+1} = \boldsymbol{\Upsilon}\Big(\mathbf{w}_n + \beta_n[\tilde{r}_n + \tilde{\gamma}\mathbf{w}_n^\top\mathbf{x}'_n - \mathbf{w}_n^\top\mathbf{x}_n]\mathbf{x}_n\Big)$	
	$= \boldsymbol{\Upsilon}\Big(\mathbf{w}_n + \beta_n[\mathbf{h}(\mathbf{w}_n) + \mathbf{N}_{n+1}]\Big)$	(4.60)
limiting ODE	$\dot{\boldsymbol{\theta}} = \mathbf{g}(\boldsymbol{\theta}) = \mathbf{b} - \mathbf{A}\boldsymbol{\theta} = -\mathbf{A}[\boldsymbol{\theta} - \boldsymbol{\theta}^*]$	(2.34)
	$\dot{\mathbf{w}} = \mathbf{h}(\boldsymbol{\theta}, \mathbf{w}) = \tilde{\mathbf{b}}(\boldsymbol{\theta}) - \tilde{\mathbf{A}}\mathbf{w} = -\tilde{\mathbf{A}}[\mathbf{w} - \mathbf{w}^*(\boldsymbol{\theta})]$	(4.55)
martingale noise	$\mathbf{M}_{n+1} = [r_n + \gamma\boldsymbol{\theta}_n^\top\mathbf{x}'_n - \boldsymbol{\theta}_n^\top\mathbf{x}_n]\mathbf{x}_n - [\mathbf{b} - \mathbf{A}\boldsymbol{\theta}_n]$	(2.37)
	$\mathbf{N}_{n+1} = [\tilde{r}_n + \tilde{\gamma}\mathbf{w}_n^\top\mathbf{x}'_n - \mathbf{w}_n^\top\mathbf{x}_n]\mathbf{x}_n - [\tilde{\mathbf{b}}(\boldsymbol{\theta}_n) - \tilde{\mathbf{A}}\mathbf{w}_n]$	(4.56)
related quantities	$\mathbf{A} = \mathbb{E}_{\pi,d_s}[\mathbf{x}_n(\mathbf{x}_n - \gamma\mathbf{x}'_n)^\top] = \mathbf{X}^\top\mathbf{D}_\pi(\mathbf{I} - \gamma\mathbf{P}_\pi)\mathbf{X}$	(2.35)
	$\mathbf{b} = \mathbb{E}_{\pi,d_s}[\mathbf{x}_n r_n] = \mathbf{X}^\top\mathbf{D}_\pi\mathbf{r}_\pi$	(2.36)
	$\tilde{\mathbf{A}} = \mathbb{E}_{\pi,d_s}[\mathbf{x}_n(\mathbf{x}_n - \tilde{\gamma}\mathbf{x}'_n)^\top] = \mathbf{X}^\top\mathbf{D}_\pi(\mathbf{I} - \tilde{\gamma}\mathbf{P}_\pi)\mathbf{X}$	(4.52)
	$\tilde{\mathbf{b}}(\boldsymbol{\theta}) = \mathbb{E}_{\pi,d_s}[\mathbf{x}_n\tilde{r}_n] = \mathbf{X}^\top\mathbf{D}_\pi\tilde{\mathbf{r}}_\pi(\boldsymbol{\theta})$	(4.53)
DV target	$\tilde{r}_n = \mathbb{E}_\pi[\delta_n^2(\boldsymbol{\theta})]$	
	$\mathbf{w}^*(\boldsymbol{\theta}) = \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{b}}(\boldsymbol{\theta})$	(4.57)
projection operator	$\boldsymbol{\Upsilon}(\mathbf{z}) = \operatorname*{argmin}_{\hat{\mathbf{z}}\in C} \|\mathbf{z} - \hat{\mathbf{z}}\|$	(4.58)

Table A.1: *A summary of the various expressions for DVTD in the ODE framework.*

Appendix B

C51 Under Linear Function Approximation

From the description of C51 (a distributional RL algorithm) provided by Bellemare, Dabney and Munos (2017), we show that the update equations actually have a fairly simple form under linear function approximation.

C51 parameterizes distributions as taking on $N+1$ possible discrete values, equally spaced between $z_{\min}$ and $z_{\max}$, with z_k defined as:

$$z_k \stackrel{\text{def}}{=} z_{\min} + k\Big(\frac{z_{\max} - z_{\min}}{N}\Big) = z_{\min} + k\Delta z, \tag{B.1}$$

with $\Delta z = (z_{\max} - z_{\min})/N$.

For every state s, it assigns a probability of the return taking on that value, denoted $p_i(s)$, with $\sum_{i=0}^{N} p_i(s) = 1$ and $p_i(s) \geq 0$ for all $i \in \{0, \ldots, N\}$ and $s \in \mathcal{S}$. Here, we are assuming linear function approximation, with a feature function $\phi : \mathcal{S} \to \mathbb{R}^K$. Let θ_i be the parameters for the i-th unit. Then $p_i(s)$ is defined as:

$$p_i(s) \stackrel{\text{def}}{=} \frac{e^{\theta_i^\top \phi(s)}}{\sum_{j=0}^{N} e^{\theta_j^\top \phi(s)}} = \frac{e^{\theta_i^\top \phi(s)}}{\sigma(s)}, \tag{B.2}$$

where:

$$\sigma(s) \stackrel{\text{def}}{=} \sum_{j=0}^{N} e^{\theta_j^\top \phi(s)}. \tag{B.3}$$

Regarding the parameters as a weight matrix $\Theta \in \mathbb{R}^{N+1 \times K}$ allows us to write:

$$\mathbf{p}(s) = \frac{1}{\sigma(s)} e^{\Theta\phi(s)} \tag{B.4}$$

C51 updates from samples of the distribution, which entails a sampled Bellman

operator $\hat{T}$. For a sample (s, r, s'), it is defined as:

$$\hat{T}z_j \stackrel{\text{def}}{=} r + \gamma z_j. \tag{B.5}$$

In order to form the loss function, C51 projects arbitrary distributions onto the supported values $\{z_0, z_1, \ldots, z_N\}$ using the Cramér projection (denoted Π_C). The projected update is denoted $\mathbf{m}(s, s') \in \mathbb{R}^{N+1}$, with $m_i(s, s')$ given by:

$$m_i(s, s') \stackrel{\text{def}}{=} [\Pi_C \hat{T} Z(s)]_i = \sum_{j=0}^{N} \left[1 - \frac{\left| [\hat{T}z_j]_{z_{\min}}^{z_{\max}} - z_i \right|}{\Delta z} \right]_0^1 p_j(s'), \tag{B.6}$$

where $[x]_a^b$ denotes that x is "clipped" to be within $[a, b]$, that is:

$$[x]_a^b \stackrel{\text{def}}{=} \min(b, \max(a, x)). \tag{B.7}$$

The loss used in (Bellemare, Dabney and Munos 2017) was the cross-entropy loss:

$$\mathcal{L} \stackrel{\text{def}}{=} \sum_{i=0}^{N} m_i(s, s') \log p_i(s). \tag{B.8}$$

Treating $m_i(s, s')$ as independent of Θ (as in TD learning) we get the semi-gradient:

$$\nabla_\Theta \mathcal{L} = -\sum_{i=0}^{N} m_i \nabla_\Theta \log p_i. \tag{B.9}$$

Hereafter we stop specifying the state (*i.e.* use m_i and p_i in place of $m_i(s, s')$ and $p_i(s)$) where the meaning is clear.

Differentiating with respect to a single weight, we get:

$$\frac{\partial \log p_i}{\partial \theta_{ab}} = \frac{\partial}{\partial \theta_{ab}} \log \left(\frac{e^{\theta_i^\top \phi}}{\sigma} \right) = \frac{\sigma}{e^{\theta_i^\top \phi}} \frac{\partial p_i}{\partial \theta_{ab}}. \tag{B.10}$$

Further differentiation reveals:

$$\frac{\partial p_i}{\partial \theta_{ab}} = \frac{\partial}{\partial \theta_{ab}} \frac{e^{\theta_i^\top \phi}}{\sigma} = \frac{1}{\sigma} \frac{\partial e^{\theta_i^\top \phi}}{\partial \theta_{ab}} - \frac{e^{\theta_i^\top \phi}}{\sigma^2} \frac{\partial \sigma}{\partial \theta_{ab}}. \tag{B.11}$$

For the first term, we have:

$$\frac{\partial e^{\theta_i^\top \phi}}{\partial \theta_{ab}} = \delta_{ia} \phi_b e^{\theta_i^\top \phi}, \tag{B.12}$$

where define δ_{ij} (the Kronecker delta) via:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{B.13}$$

Evaluating the second term gives:

$$\frac{\partial \sigma}{\partial \theta_{ab}} = \frac{\partial}{\partial \theta_{ab}} \sum_j e^{\theta_j^\top \phi} = \sum_j \frac{\partial}{\partial \theta_{ab}} e^{\theta_j^\top \phi} = \sum_j \delta_{ja} \phi_b e^{\theta_j^\top \phi} = \phi_b e^{\theta_a^\top \phi}, \tag{B.14}$$

so altogether we have:

$$\begin{aligned} \frac{\partial p_i}{\partial \theta_{ab}} &= \frac{1}{\sigma} \frac{\partial e^{\theta_i^\top \phi}}{\partial \theta_{ab}} - \frac{e^{\theta_i^\top \phi}}{\sigma^2} \frac{\partial \sigma}{\partial \theta_{ab}} = \frac{\delta_{ia} \phi_b e^{\theta_i^\top \phi}}{\sigma} - \frac{\phi_b e^{\theta_i^\top \phi} e^{\theta_a^\top \phi}}{\sigma^2} \\ &= \frac{1}{\sigma^2} \phi_b e^{\theta_i^\top \phi} (\delta_{ia} \sigma - e^{\theta_a^\top \phi}) \end{aligned} \tag{B.15}$$

Returning to the original derivative, we get:

$$\begin{aligned} \frac{\partial \log p_i}{\partial \theta_{ab}} &= \frac{\sigma}{e^{\theta_i^\top \phi}} \frac{\partial p_i}{\partial \theta_{ab}} = \left(\frac{\sigma}{e^{\theta_i^\top \phi}} \right) \left(\frac{1}{\sigma^2} \phi_b e^{\theta_i^\top \phi} \left(\delta_{ia} \sigma - e^{\theta_a^\top \phi} \right) \right) \\ &= \frac{1}{\sigma} \phi_b (\delta_{ia} \sigma - e^{\theta_a^\top \phi}) \end{aligned} \tag{B.16}$$

Now, to compute the gradient for the loss, we have to take the sum:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta_{ab}} &= -\sum_{i=0}^{N} m_i \frac{\partial \log p_i}{\partial \theta_{ab}} = -\sum_{i=0}^{N} \frac{m_i \phi_b}{\sigma} (\delta_{ia} \sigma - e^{\theta_a^\top \phi}) \\ &= \phi_b \left(\frac{e^{\theta_a^\top \phi}}{\sigma} \sum_{i=0}^{N} m_i - \sum_{i=0}^{N} m_i \delta_{ia} \right) \\ &= \phi_b \left(p_a \cancel{\sum_i m_i}^{1} - m_a \right) \\ &= \phi_b (p_a - m_a), \end{aligned} \tag{B.17}$$

where we use the fact that the m_i sum to one, since $\mathbf{m}$ represents a probability distribution. The matrix version of the above is just:

$$\nabla_\theta \mathcal{L} = (\mathbf{p} - \mathbf{m}) \phi^\top. \tag{B.18}$$

Appendix C

Algorithm Listing

Algorithm 1: Linear TD(λ)

Linear TD(λ) learns a value function for a policy π using linear function approximation. Its value function is $\hat{v} : \mathcal{S} \to \mathbb{R}$ with $\hat{v} = \boldsymbol{\theta}^\top \phi(s)$, where $\phi : \mathcal{S} \to \mathbb{R}^n$ is the function that maps states to features. As the agent learns, its value function becomes closer to the true value of the policy, that is, $\hat{v}(s) \approx v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$

initialize(π, ϕ, α, γ, λ):
- ▷ $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the policy, assigns probabilities to state-action pairs
- ▷ $\phi : \mathcal{S} \to \mathbb{R}^n$ is the representation, mapping states to features
- ▷ α, the learning rate, either constant or specified as a sequence $\{\alpha\}_t$
- ▷ $\gamma : \mathcal{S} \to [0, 1]$ is the discount function
- ▷ $\lambda : \mathcal{S} \to [0, 1]$ is the bootstrapping function
- Initialize $\boldsymbol{\theta} \in \mathbb{R}^n$ arbitrarily
- Allocate eligibility trace vector $\mathbf{z} \in \mathbb{R}^n$

update(s, a, r, s'):
- $\mathbf{x} \leftarrow \phi(s)$
- $\mathbf{x}' \leftarrow \phi(s')$ ▷ compute features
- $\delta \leftarrow r + \gamma(s')\boldsymbol{\theta}^\top \mathbf{x}' - \boldsymbol{\theta}^\top \mathbf{x}$ ▷ temporal difference error
- $\mathbf{z} \leftarrow \gamma(s')\lambda(s')\mathbf{z} + \mathbf{x}$ ▷ update eligibility trace
- $\boldsymbol{\theta} \leftarrow \alpha\delta\mathbf{z}$ ▷ update weight vector

reset():
- Set $\mathbf{z} \in \leftarrow \mathbf{0}$, for $\mathbf{z} \in \mathbb{R}^n$ ▷ clear eligibility traces

learn(*episodes*):
- **for** each episode **do**
  - reset() ▷ Get ready for episode start
  - Get s, the start state for the episode
  - **while** s is not *terminal* **do**
    - Choose $a \sim \pi(\cdot|s)$
    - Execute action a, get reward r and subsequent state s'
    - update(s, a, r, s')
    - Set $s \leftarrow s'$ ▷ prepare for next iteration
- **return** $\hat{v} : s \mapsto \boldsymbol{\theta}^\top \phi(s)$

Algorithm 2: Linear DVTD(λ)

Linear DVTD(λ) learns a value function ($\hat{v} : \mathcal{S} \to \mathbb{R}$) *and* a variance function ($\hat{u} : \mathcal{S} \to \mathbb{R}$) for a policy π using linear function approximation. As the agent learns, its value function becomes closer to the true value function; as the value function becomes more accurate ($\hat{v} \approx \mathbb{E}_\pi[G_t^\lambda | S_t = s]$), the variance function becomes a better approximation of the variance of the λ-return, *i.e.* $\hat{u}(s) \approx \mathbb{E}_\pi[(G_t^\lambda - \hat{v}(s))^2 | S_t = s]$.

initialize(π, ϕ, α, $\tilde{\alpha}$, γ, λ, $\tilde{\lambda}$):
- ▷ $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the policy, assigns probabilities to state-action pairs
- ▷ $\phi : \mathcal{S} \to \mathbb{R}^n$ is the representation, mapping states to features
- ▷ α, value learning rate, either constant or specified as a sequence $\{\alpha\}_t$
- ▷ $\tilde{\alpha}$, variance learning rate, either constant or specified as a sequence $\{\tilde{\alpha}\}_t$
- ▷ $\gamma : \mathcal{S} \to [0, 1]$ is the discount function
- ▷ $\lambda : \mathcal{S} \to [0, 1]$ is the bootstrapping function for the value learner
- ▷ $\tilde{\lambda} : \mathcal{S} \to [0, 1]$ is the bootstrapping function for the variance learner
- Initialize $\mathbf{\theta} \in \mathbb{R}^n$ arbitrarily ▷ value estimator weights
- Initialize $\mathbf{w} \in \mathbb{R}^n$ arbitrarily ▷ variance estimator weights

update(s, a, r, s'):
- $\mathbf{x} \leftarrow \phi(s)$
- $\mathbf{x}' \leftarrow \phi(s')$ ▷ compute features
- $\delta \leftarrow r + \gamma(s')\mathbf{\theta}^\top \mathbf{x}' - \mathbf{\theta}^\top \mathbf{x}$ ▷ temporal difference error
- $\mathbf{z} \leftarrow \gamma(s')\lambda(s')\mathbf{z} + \mathbf{x}$ ▷ update eligibility trace
- $\mathbf{\theta} \leftarrow \alpha\delta\mathbf{z}$ ▷ update weight vector for $\hat{v}$
- $\tilde{r} \leftarrow \delta^2$ ▷ "reward" for DV component
- $\tilde{\gamma} \leftarrow (\gamma(s')\lambda(s'))^2$
- $\tilde{\delta} \leftarrow \tilde{r} + \tilde{\gamma}\mathbf{w}^\top \mathbf{x}' - \mathbf{w}^\top \mathbf{x}$
- $\tilde{\mathbf{z}} \leftarrow \tilde{\gamma}\tilde{\lambda}(s')\tilde{\mathbf{z}} + \mathbf{x}$
- $\mathbf{w} \leftarrow \tilde{\alpha}\tilde{\delta}\tilde{\mathbf{z}}$ ▷ update weight vector for variance

reset():
- Set $\mathbf{z} \in\leftarrow \mathbf{0}$, for $\mathbf{z} \in \mathbb{R}^n$
- Set $\tilde{\mathbf{z}} \leftarrow \mathbf{0}$ for $\tilde{\mathbf{z}} \in \mathbb{R}^n$. ▷ clear eligibility traces

learn(*episodes*):
- **for** each episode **do**
  - reset() ▷ Get ready for episode start
  - Get s, the start state for the episode
  - **while** s is not *terminal* **do**
    - Choose $a \sim \pi(\cdot|s)$
    - Execute action a, get reward r and subsequent state s'
    - update(s, a, r, s')
    - Set $s \leftarrow s'$ ▷ prepare for next iteration
- **return** $\hat{v} : s \mapsto \mathbf{\theta}^\top \phi(s)$

Algorithm 3: Linear VTD(λ)

Linear VTD(λ) learns a value function, $\hat{v} : \mathcal{S} \to \mathbb{R}$, and an estimate for the second moment, $\hat{v}^{(2)} : \mathcal{S} \to \mathbb{R}$. As the agent learns, its value function better approximates the λ-return, so that $\hat{v}(s) \to \mathbb{E}_\pi[G_t^\lambda | S_t = s]$, and $\hat{v}^{(2)}$ goes to $\hat{v}^{(2)}(s) \approx \mathbb{E}_\pi[(G_t^\lambda)^2 | S_t = s]$.

initialize(π, ϕ, α, $\tilde{\alpha}$, γ, λ, $\tilde{\lambda}$):
- ▷ $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the policy, assigns probabilities to state-action pairs
- ▷ $\phi : \mathcal{S} \to \mathbb{R}^n$ is the representation, mapping states to features
- ▷ α, value learning rate, either constant or specified as a sequence $\{\alpha\}_t$
- ▷ $\tilde{\alpha}$, SM learning rate, either constant or specified as a sequence $\{\tilde{\alpha}\}_t$
- ▷ $\gamma : \mathcal{S} \to [0, 1]$ is the discount function
- ▷ $\lambda : \mathcal{S} \to [0, 1]$ is the bootstrapping function for the value learner
- ▷ $\tilde{\lambda} : \mathcal{S} \to [0, 1]$ is the bootstrapping function for the second moment learner
- Initialize $\mathbf{\theta} \in \mathbb{R}^n$ arbitrarily ▷ value estimator weights
- Initialize $\mathbf{w} \in \mathbb{R}^n$ arbitrarily ▷ second moment estimator weights

update(s, a, r, s'):
- $\mathbf{x} \leftarrow \phi(s)$
- $\mathbf{x}' \leftarrow \phi(s')$ ▷ compute features
- $\delta \leftarrow r + \gamma(s')\mathbf{\theta}^\top \mathbf{x}' - \mathbf{\theta}^\top \mathbf{x}$ ▷ temporal difference error
- $\mathbf{z} \leftarrow \gamma(s')\lambda(s')\mathbf{z} + \mathbf{x}$ ▷ update eligibility trace
- $\mathbf{\theta} \leftarrow \alpha\delta\mathbf{z}$ ▷ update weight vector for $\hat{v}$
- $\tilde{r} \leftarrow r^2 + 2\gamma(s')r\mathbf{\theta}^\top \mathbf{x}'$ ▷ "reward" for SM component
- $\tilde{\gamma} \leftarrow (\gamma(s')\lambda(s'))^2$
- $\tilde{\delta} \leftarrow \tilde{r} + \tilde{\gamma}\mathbf{w}^\top \mathbf{x}' - \mathbf{w}^\top \mathbf{x}$
- $\tilde{\mathbf{z}} \leftarrow \tilde{\gamma}\tilde{\lambda}(s')\tilde{\mathbf{z}} + \mathbf{x}$
- $\mathbf{w} \leftarrow \tilde{\alpha}\tilde{\delta}\tilde{\mathbf{z}}$ ▷ update weight vector for variance

reset():
- Set $\mathbf{z} \in \leftarrow \mathbf{0}$, for $\mathbf{z} \in \mathbb{R}^n$ Set $\tilde{\mathbf{z}} \leftarrow \mathbf{0}$ for $\tilde{\mathbf{z}} \in \mathbb{R}^n$. ▷ clear eligibility traces

learn(*episodes*):
- **for** each episode **do**
  - reset() ▷ Get ready for episode start
  - Get s, the start state for the episode
  - **while** s is not *terminal* **do**
    - Choose $a \sim \pi(\cdot|s)$
    - Execute action a, get reward r and subsequent state s'
    - update(s, a, r, s')
    - Set $s \leftarrow s'$ ▷ prepare for next iteration
- **return** $\hat{v}^{(2)} : s \mapsto \mathbf{w}^\top \phi(s) - (\mathbf{\theta}^\top \phi(s))^2$

Algorithm 4: Moment TD(λ)

input: the policy π to be evaluated
input: a set of n differentiable functions $\{v_i\}_{i=1}^n$ parameterized by their respective weights $\{\mathbf{w}\}_{i=1}^n$ with $\mathbf{w}_i \in \mathbb{R}^d$, such that $v_i : \mathcal{S} \to \mathbb{R}$ and $v_i(\text{terminal}) = 0$.
parameter: a set of stepsizes $\{\alpha_i\}_{i=1}^n$ with $\alpha_i \in (0, 1)$;
parameter: a set of trace decay rates $\{\lambda\}_{i=1}^n$ with $\lambda_i \in [0, 1]$.

foreach $k = 1, \dots, n$ **do** Initialize $\mathbf{w}_k$ arbitrarily
for each episode **do**
    Initialize s ▷ Initial state
    foreach $k = 1, \dots, n$ **do** $\mathbf{z}_k \leftarrow \mathbf{0}$ set traces to zero
    while s is not terminal **do**
        Choose $a \sim \pi(\cdot|s)$
        Take action a, observe r, s'
        for $k = n, \dots, 1$ **do**
            $r_k \leftarrow \sum_{\ell=0}^{k-1} \binom{k}{\ell} \gamma^\ell r^{k-\ell} v_\ell(s')$ ▷ "reward" from (3.25)
            $\mathbf{z}_k \leftarrow \gamma^k \lambda_k + \nabla v_k(s)$
            $\delta \leftarrow r_k + \gamma^k v_k(s') - v_k(s)$
            $\mathbf{w}_k \leftarrow \mathbf{w}_k + \alpha_k \delta \mathbf{z}_k$
        $s \leftarrow s'$

Algorithm 5: Linear C51 (Categorical Distributional Reinforcement Learning)

The C51 algorithm learns to assign probabilities to returns, rather than just learning the expected value of the return. It does this by parameterizing the return distribution as taking on K possible values, equally spaced between $Z_{\min}$ and $Z_{\min}$. With each transition sample (s, a, r, s'), the algorithm samples from the (approximate) distribution of future returns using, projects it onto the categorical support, and then updates the estimate of the distribution of s by comparing the two. The approximate distribution can then be used to estimate the expected return or its variance, among other things.

We have modified the algorithm described in Bellemare, Dabney and Munos 2017 (Algorithm 1, pg. 6) to be more self-contained and to explicitly use linear function approximation.

initialize(π, ϕ, α, γ, K, $Z_{\min}$, $Z_{\max}$):
- $\triangleright$ $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the policy, assigns probabilities to state-action pairs
- $\triangleright$ $\mathbf{x} : \mathcal{S} \to \mathbb{R}^n$ is the representation, mapping states to features
- $\triangleright$ α, learning rate, either constant or specified as a sequence $\{\alpha\}_t$
- $\triangleright$ $\gamma : \mathcal{S} \to [0, 1]$ is the discount function
- $\triangleright$ $K \in \mathbb{N}_+$ is the number of "atoms" in the categorical distribution
- $\triangleright$ $Z_{\min}$ is the minimum value the categorical distribution supports
- $\triangleright$ $Z_{\max}$ is the maximum value the categorical distribution supports

  Initialize $\mathbf{W} \in \mathbb{R}^{K \times n}$ arbitrarily, e.g. to all ones. $\triangleright$ distribution estimator weights

  Set $\Delta z \leftarrow \left(\frac{Z_{\max} - Z_{\min}}{K}\right)$ $\triangleright$ spacing between atoms

  Initialize $\mathbf{z} \in \mathbb{R}^K$ such that $z_i = Z_{\min} + (i-1)\Delta z$ $\triangleright$ distribution support

compute_distribution(s):

  $\mathbf{a} \leftarrow \exp\{\mathbf{W}\mathbf{x}(s)\}$

  return $\frac{\mathbf{a}}{\|\mathbf{a}\|_1}$ $\triangleright$ vector sigmoid function

update(s, a, r, s'):

  $\mathbf{p} \leftarrow$ compute_distribution(s)

  $\mathbf{p}' \leftarrow$ compute_distribution(s') $\triangleright$ get distribution estimates for s and s'

  $\mathcal{T}\mathbf{z} \leftarrow [r + \gamma(s')\mathbf{z}]_{Z_{\min}}^{Z_{\max}}$ $\triangleright$ sample (clipped) distributional Bellman operator

  $\mathbf{m} \leftarrow \mathbf{0}$ $\triangleright$ $\mathbf{m} \in \mathbb{R}^K$ will be $\mathbf{p}'$ projected onto the supported values

  for $i = 1, 2, \ldots, K$ **do**

    for $j = 1, 2, \ldots, K$ **do**

      $m_i \leftarrow m_i + p'_j \left[1 - \frac{|[\mathcal{T}z]_j|}{\Delta z}\right]_0^1$

  $\mathbf{W} \leftarrow \mathbf{W} - \alpha(\mathbf{p} - \mathbf{m})\mathbf{x}^\top$ $\triangleright$ update weights

learn(*episodes*):

  for each episode **do**

    Get s, the start state for the episode

    while s is not *terminal* **do**

      Choose $a \sim \pi(\cdot|s)$

      Execute action a, get reward r and subsequent state s'

      update(s, a, r, s')

      Set $s \leftarrow s'$ $\triangleright$ prepare for next iteration

  return $\hat{z} : s \mapsto \text{Dist}(\mathbf{z})$ $\triangleright$ A function mapping states to distributions

Appendix D

Glossary

action A response an agent could execute in response to a given state which affects the environment.

action space, set of actions The set of all possible actions that could be executed in an environment, denoted $\mathcal{A}$.

agent, learning agent An entity (*e.g.*, a computer program) that interacts with an environment in a learning task.

approximation target The quantity a learning algorithm estimates. For example, a least squares estimator ($\mathbf{w} = \operatorname{argmin}_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$) targets $\mathbf{y}$ while the approximation target for TD(λ) is the λ-return.

Bellman equation A recursive equation that expresses the value of the current state in terms of the reward from the next transition and the (discounted) value of the subsequent state. For example, $v(s) = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1})|S_t = s]$ is a Bellman equation; however usually such equations are with reference to a particular policy (particularly the *optimal* policy).

Bellman operator An operator that transforms value functions. In this thesis we concern ourselves mainly with the Bellman policy operator, expressed as $\mathcal{T}^\pi \mathbf{v} \stackrel{\text{def}}{=} \mathbf{r}_\pi + \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{v}$, where π is some policy. Applying $\mathcal{T}^\pi$ to $\mathbf{v}$ yields a new value function, comprised of the expected reward and the discounted value of the next state (according to $\mathbf{v}$), given that we select actions according to π. The Bellman operator is a contraction; which is to say if we update our value function like $\mathbf{v}_{t+1} = \mathcal{T}^\pi \mathbf{v}_t$, then $\lim_{t\to\infty} \mathbf{v}_t = \mathbf{v}_\pi$, which is the fixed-point of $\mathcal{T}^\pi$ and (not coincidentally) also the value function for policy π.

bootstrapping factor A tunable parameter, usually denoted λ, that controls the degree by which an algorithm bootstraps from its current value function instead of the full return. For TD(λ), this is implemented as the decay rate for the eligibility trace. It can be state-dependent, in which case we write $\lambda(s)$ for the value of λ in state $s \in \mathcal{S}$.

central moment A measure of how much of a random variable's probability mass is concentrated over values distant from the mean. If μ is the mean of a random variable, then the n-th central moment is defined as:

$$c_n = \begin{cases} \int_{\mathcal{X}} p(x)(x-\mu)^n \mathrm{d}x & \text{continuous case} \\ \sum_{x\in\mathcal{X}} p(x)(x-\mu)^n & \text{discrete case} \end{cases}$$

For example, the second central moment (also known as the *variance*) of a random variable X is written $c_2(X)$.

continuing setting A problem where the environment doesn't have terminal states and trajectories can continue for arbitrary lengths of time.

contraction, operator contraction For a function, mapping, or operator (say $f : \mathcal{X} \to \mathcal{X}$ for some normed vector space $\mathcal{X}$) to be a contraction (alternatively: contracting, contractive) means that in some norm, $\|\cdot\|$, we have $\|f(x)\| < \|x\|$ for all $x \in \mathcal{X}$. Showing that an operator is a contraction is useful because then it probably also has a fixed-point. More general definitions of contractions exist[1], *e.g.*, for metric spaces instead of just normed vector spaces, but this definition suffices for our purposes.

convergent A sequence that tends to grow closer to a particular point is said to be "convergent". There are many different kinds of convergence, but for stochastic approximation schemes we usually mean that given enough time, its estimates will eventually reach a fixed point regardless of the initial conditions.

differential equation An equation where the derivative of a function is defined with reference to the function itself. There are many kinds of differential equations, although for our purposes we mainly consider *ordinary* differential equations.

discount factor A scalar between zero and one that expresses how much the value of a state should influence the value of its predecessor states, usually denoted by γ. In a slight departure from the usual MDP formalism, we may consider state- or transition-dependent discounting, where $\gamma_{t+1} = \gamma(S_{t+1})$ or $\gamma_{t+1} = \gamma(S_t, A_t, S_{t+1})$. Such considerations are an element of the theory for *general value functions*.

DVTD, Direct Variance Temporal Difference Learning Direct Variance Temporal Difference learning, which is a method for estimating the variance of the return using temporal difference errors (and a major component of this very thesis).

dynamic programming A collection of techniques for efficiently solving problems by exploiting the structure of the task, *e.g.* by separating it into smaller subproblems and tackling those individually, or amortizing resource use by constructing solutions incrementally. Reinforcement learning has a large amount of overlap with dynamic programming; for example, TD(λ) can be thought of as a dynamic programming algorithm due to the way it estimates the value function by bootstrapping off existing estimates.

environment Everything external to the agent in a learning task.

episodic setting A task where trajectories can be split into discrete *episodes*, separated when the environment reaches a terminal state. For example, the game of chess is an episodic problem, and each separate game can be thought of as an episode.

[1]Blame the topologists for always poking holes in things

fast timescale For a stochastic approximation scheme with components that update with different stepsizes, the fast timescale corresponds to the component whose stepsize dominates that of the "slow" component. From the perspective of the fast component, the slow component is quasi-stationary; this perspective helps considerably when considering questions of convergence and stability. See also *slow timescale*.

filtration A representation of the information available with regards to a stochastic process at a given time. For example, the history of states visited in an MDP corresponds to a filtration.

fixed-point The fixed-point (sometimes written fixed point) of a sequence is the value that the sequence ultimately converges to.

function approximation A problem setting where the value function is parameterized and therefore only represented approximately.

general value function, GVF An extension to the theory of value function to encompass more general predictions, for example those that incorporate state-dependent discounting or bootstrapping. More general development of GVFs can be found in Sutton, Modayil et al. 2011; White 2015; Modayil, White and Sutton 2012; Sherstan 2020, among others.

Hermitian matrix A (square) matrix that is unchanged under the conjugate transpose, that is, $\mathbf{A} = \mathbf{A}^\dagger$,. Note that a symmetric real matrix is trivially Hermitian.

interpolated differential equation A construction used to make discrete time iterates comparable with continuous time functions, *e.g.* to facilitate comparisons between a stochastic approximation scheme and its associated limiting ODE.

learning task A task or problem which we pose to a learning agent. In reinforcement learning, we usually we model the task as an MDP.

Liapunov function, Liapunov method (also romanized as "Lyapunov") Liapunov's method for showing the asymptotic stability of an ODE entails selecting a function that effectively acts as a "potential" (think of gravity). For example, suppose $V(\cdot)$ is a Liapunov function for the system $\dot{x}(t) = f(x)$. Then V is positive, except at the equilibrium, with continuous first derivatives, and furthermore, $\frac{\mathrm{d}}{\mathrm{d}t}V = \langle \nabla V, \dot{x} \rangle = \langle \nabla V, f \rangle$ is negative definite. The existence of such a function demonstrates that (at least near the equilibrium point), the system is always getting closer to the equilibrium; therefore it will converge to the equilibrium and stay there.

limiting differential equation The differential equation associated with a stochastic approximation scheme in the limit as the stepsize (and noise) approach zero.

linear function approximation A problem setting where the value function is parameterized in such a way that a state's value can be expressed as the inner product of the feature vector and the weight vector. For example, if $\phi(s_t) = \phi_t$ is the feature vector at time t, and θ is the weight vector, then $v(s_t) = \theta^\top \phi_t$.

Markov decision problem, Markov decision process, *MDP* A Markov Decision Problem is construction which we use to model various tasks. See Chapter 2 for more details.

Markov property The defining characteristic of Markov processes; essentially states that the current state contains all relevant information for predicting the process' future. A Markov process is "memoryless" in that the past history does not influence the future evolution.

Markov reward process, *MRP* A stochastic process with the Markov property, *i.e.* it is *memoryless* in that knowing the current state is just as informative as knowing the entire history of previous states. A Markov reward process is induced from an MDP by selecting actions according to a fixed policy.

martingale A stochastic process $\{X_t\}$ that where the expected value of the next observation, conditioned on all past observations, is equal to the most recent observation. That is, $\mathbb{E}[X_{n+1}|X_1, \ldots, X_n] = X_n$.

martingale difference sequence, MDS A sequence $\{M_t\}$ is a martingale difference sequence if its expectation with respect to the past is zero. More formally, let $\{M_t\}$ be a martingale difference sequence with respect to the sequence of filtrations $\{\mathcal{F}_t\}$. Then for all t, we have $\mathbb{E}|M_t| < \infty$ and $\mathbb{E}[M_{t+1}|\mathcal{F}_t] = 0$. Note that if X_t is a martingale, then $M_t = X_t - X_{t-1}$ is a martingale difference sequence (hence the name).

moment of a random variable A measure of the probability mass of a random variable. More concretely, the n-th moment of a random variable with probability distribution function $p(\cdot)$ is defined as

$$\mu_n = \begin{cases} \int_{\mathcal{X}} p(x) x^n \mathrm{d}x & \text{continuous case} \\ \sum_{x \in \mathcal{X}} p(x) x^n & \text{discrete case} \end{cases}$$

For example, the mean of a random variable X is $\mu_1(X)$, the first moment.

multiple timescale algorithm A stochastic approximation scheme with multiple components that update at different rates, particularly those with multiple stepsizes.

objective function, error function A function used to formalize how "good" a particular solution is for a given problem. For example, $f(x) = (x - x^*)^2$ is an objective function (the squared error) that measures the distance between x and x^*. It is generally helpful if the objective function is differentiable (or something like it), since that allows us to improve our solution by gradient descent.

observation A term for the agent's perception of the environment's state. In the tabular case, the observations are equivalent to the state, whereas under function approximation (or in partially observable MDPs) the agent may not have access to the "true" state information, so we refer to the sensory information it receives as observations when a distinction may be required.

ordinary differential equation (ODE) An equation where the derivative of a function is defined with reference to the function itself. For example, $\ddot{x} = \omega^2 x$, the equation for the undamped harmonic oscillator, is an ODE.

policy A method of selecting actions, usually denoted π. Can be stochastic, in which case the probability of selecting $a \in \mathcal{A}$ given state $s \in \mathcal{S}$ is usually written $\pi(a|s)$.

positive definite matrix A matrix $\mathbf{A}$ such that $\mathbf{x}^\top \mathbf{A}\mathbf{x} > 0$ for $\mathbf{x} \neq 0$. If we relax the condition to $\mathbf{x}^\top \mathbf{A}\mathbf{x} \geq 0$, we say that $\mathbf{A}$ is positive semi-definite instead.

raw moment See *Moment of a random variable*

real matrix A matrix with real-valued entries, e.g. $\mathbf{A} \in \mathbb{R}^{m \times n}$.

return The sum of discounted rewards starting from a point in time. Usually denoted G_t, and defined $G_t = \sum_{n=0}^{\infty} R_{t+n+1} \prod_{k=1}^{n-1} \gamma_{t+k}$, or just $G_t = \sum_{n=0} \gamma^n R_{t+n+1}$ for constant discounting.

return error, RE, $\overline{\text{RE}}$ The error of some value function with respect to the return under a given policy and state distribution. Usually the metric used is the L_2-norm, yielding the mean squared return error.

reward A scalar signal emitted by the environment as part of a transition.

reward function A function that returns a reward for a given transition, potentially with an element of randomness or noise. Formally, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{R} \to [0, 1]$.

sigma algebra, σ-algebra A sigma algebra or σ-algebra is a collection of subsets of some set (say X) that includes X and is closed under both complement and countable unions. They tend to be invoked when something requires Lebesgue integration, which is rather often in probability and statistics.

slow timescale For a coupled stochastic approximation scheme with components that update with different stepsizes, the slow timescale is the one whose stepsize is dominated by the stepsize of the "fast" component. If we identify the sum of the stepsizes with the time elapsed (*e.g.* $t_n = \sum_{k=0} \alpha_k$) then the fast component, with larger stepsizes relative to the slow component, effectively has progressed further in time compared to the slow component (for the same number of iterations).

stable, stability The tendency of a system to be robust to small perturbations. For a stochastic approximation scheme, we say that it is stable if there is no risk that it will diverge (go to infinity) under reasonable conditions.

state A configuration of the environment.

state space, set of states The set of all possible configurations of an environment, denoted $\mathcal{S}$.

stepsize A parameter that determines the magnitude of the updates made to the value function. Large stepsizes may engender quicker learning, at the expense of greater instability and with the possibility of "forgetting" past experience.

symmetric matrix A (square) matrix such that $\mathbf{A} = \mathbf{A}^\top$.

tabular case A problem setting where the observations available to the agent express the full state information. This means that the value function is essentially a table assigning states to values, thus we describe such tasks as being "tabular" in nature.

temporal difference error, TD error A measure of the error in the value function for a single transition. Formally, $\delta_t = R_{t+1} + \gamma_{t+1} v(S_{t+1}) - v(S_t)$. It is the difference between the current state's value and the discounted value of the successor state plus the reward for that transition.

terminal state A state in a Markov process which always transitions to itself and returns a reward of zero for every transition. Reaching the terminal state implies that the trajectory is over; in the episodic case this corresponds to the end of an episode.

transition A grouping of state, action, subsequent state (and possibly the reward) for a given time-step. Formally, a tuple (s, a, s') or (s, a, r, s'), where s and s' are the current state and successor state, respectively, a is the action executed in state s, and r is the reward produced by the transition.

transition function A function that provides the probability of making a particular transition. For example, given a state, an action, and a successor state, the transition function returns the probability of making that particular transition. Formally, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. To be well-defined, we require that $\sum_{s'} P(s, a, s') = 1 \, \forall s \in \mathcal{S}, a \in \mathcal{A}$, that is the transition probabilities given state s and action a must sum to one.

Conditioned on a policy, we may instead write $\mathbf{P}_\pi$ to denote a transition matrix, whose (i,j)-th entry marks the probability of transitioning to state j given the agent is currently in state i and acting according to policy π.

update equation An equation that describes how to adjust the value function (or the weight parameters that determine the value function) in light of new experience.

value The expected return (for a given state), usually defined to be conditional on following a particular policy. When unqualified, we typically mean the true expected value, rather than an approximation or estimate of it.

value error, VE, $\overline{\text{VE}}$ The error of some value function with respect to the true value function for some policy, taken over a given state distribution. Usually the metric used is the L_2-norm, yielding the mean squared value error.

VTD, Variance Temporal Difference Learning Variance Temporal Difference learning, a method for estimating the variance of the return by first estimating the return's second moment.

σ-algebra See *sigma algebra*

D.1 Abbreviations

MDP Markov decision process

MSE mean squared error

MSVE mean squared value error

MSPBE mean squared projected Bellman error

MSTDE mean squared temporal difference error

MSRE mean squared return error

RBF radial basis function

RL reinforcement learning

RMSE root mean squared error

RMSVE root mean squared value error

TD temporal difference

DV direct variance

DVTD direct variance temporal difference learning

VTD variance temporal difference learning

LASE Locally asymptotically stable equilibrium

MC Monte Carlo

ODE Ordinary differential equation

DE Differential equation

DP Dynamic Programming

OLS Ordinary least squares

Bibliography

Bellemare, M. G., W. Dabney and R. Munos (2017). 'A Distributional Perspective on Reinforcement Learning'. *arXiv preprint*. arXiv: 1707.06887. 43, 44, 47, 110, 111, 117

Bellemare, M. G., N. Le Roux, P. S. Castro and S. Moitra (2019). 'Distributional Reinforcement Learning with Linear Function Approximation'. In: *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*. Okinawa. 43

Bennett, B., W. Chung, M. Zaheer and V. Liu (2019). 'Incrementally Learning Functions of the Return'. *arXiv preprint*. arXiv: 1907.04651. 39

Bertsekas, D. P. (2012). *Dynamic Programming and Optimal Control*. 4th ed. Vol. 2. Athena Scientific. 20

Bhandari, J., D. Russo and R. Singal (2018). 'A Finite Time Analysis of Temporal Difference Learning With Linear Function Approximation'. *arXiv preprint*. arXiv: 1806.02450. 66

Borkar, V. S. (2008). *Stochastic Approximation: A Dynamical Systems Viewpoint*. 1st ed. Cambridge University Press. 22, 24, 101

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba (2016). 'OpenAI Gym'. *arXiv preprint*. arXiv: 1606.01540. 83

Dalal, G., B. Szörényi, G. Thoppe and S. Mannor (2017). 'Finite Sample Analyses for TD(0) with Function Approximation'. *arXiv preprint*. arXiv: 1704.01161. 19, 66

David, H. A. (1995). 'First (?) Occurrence of Common Terms in Mathematical Statistics'. *The American Statistician* 49.2, pp. 121–133. 32

Ghiassian, S., H. Yu, B. Rafiee and R. S. Sutton (2018). 'Two Geometric Input Transformation Methods for Fast Online Reinforcement Learning with Neural Nets'. *arXiv preprint*. arXiv: 1805.07476. 19

Hirsch, M. W., S. Smale and R. L. Devaney (2013). *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Elsevier. 20

Jain, A., K. Khetarpal and D. Precup (2018). 'Safe Option-Critic: Learning Safety in the Option-Critic Architecture'. *arXiv preprint*. arXiv: 1807.08060. 97

Kretchmar, R. M. and C. W. Anderson (1997). 'Comparison of CMACs and Radial Basis Functions for Local Function Approximators in Reinforcement Learning'. In: *Proceedings of International Conference on Neural Networks (ICNN'97)*. Vol. 2. IEEE, pp. 834–837. 84

Levin, D. and Y. Peres (2017). *Markov Chains and Mixing Times*. 2nd ed. Providence, Rhode Island: American Mathematical Society, p. 447. 19

Lyle, C., M. G. Bellemare and P. S. Castro (2019). 'A Comparative Analysis of Expected and Distributional Reinforcement Learning'. *Proceedings of the AAAI Conference on Artificial Intelligence* 33, pp. 4504–4511. 43

Maei, H. R. (2011). 'Gradient Temporal-Difference Learning Algorithms'. PhD thesis. University of Alberta. 7, 58

Mahmood, A. (2017). 'Incremental Off-policy Reinforcement Learning Algorithms'. PhD thesis. University of Alberta. 7, 14, 16, 63

Metropolis, N. (1987). 'The Beginning of the Monte Carlo Method'. *Los Alamos Science* 15, pp. 125–130. 10

Modayil, J., A. White and R. S. Sutton (2012). 'Multi-timescale Nexting in a Reinforcement Learning Robot'. In: *From Animals to Animats 12*. Ed. by T. Ziemke, C. Balkenius and J. Hallam. Springer Berlin Heidelberg, pp. 299–309. arXiv: arXiv:1112.1133v3. 121

Munos, R. and A. Moore (1999). 'Influence and Variance of a Markov Chain: Application to Adaptive Discretization in Optimal Control'. In: *IEEE Conference on Decision and Control*, pp. 1464–1469. 32, 38

Nagurney, A. and D. Zhang (1996). *Projected Dynamical Systems and Variational Inequalities with Applications*. 1st ed. Springer US. 102

Pearson, K. (1893). 'Asymmetrical Frequency Curves'. *Nature* 49.1253, p. 6. 32

Press, W. H., S. A. Teukolsky, W. T. Vetterling and B. P. Flannery (2007). *Numerical Recipes: The Art of Scientific Computing*. 3rd ed. Cambridge University Press. 17

Recorde, R., J. Dee, R. Norton, J. Mellis and R. N. Gent (1618). *The Ground of Arts: Teaching the Perfect Worke and Practise of Arithmeticke*. I. B.

Rowland, M., M. G. Bellemare, W. Dabney, R. Munos and Y. W. Teh (2018). 'An Analysis of Categorical Distributional Reinforcement Learning'. In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. Vol. 84. Proceedings of Machine Learning Research. Lanzarote: PMLR, pp. 29–37. arXiv: 1802.08163. 44

Sato, M., H. Kimura and S. Kobayashi (2002). 'TD Algorithm for the Variance of Return and Mean-Variance Reinforcement Learning'. *Transactions of the Japanese Society for Artificial Intelligence* 16.3, pp. 353–362. 32, 38

Scherrer, B. (2010). 'Should One Compute the Temporal Difference Fix Point or Minimize the Bellman Residual? The Unified Oblique Projection View'. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa: Omnipress, pp. 959–966. arXiv: 1011.4362. 58

Sherstan, C. (2020). 'Representation and General Value Functions'. PhD thesis. University of Alberta. 121

Sherstan, C., B. Bennett, K. Young, D. R. Ashley, A. White, M. White and R. S. Sutton (2018). 'Directly Estimating the Variance of the λ-Return Using Temporal-Difference Methods'. *arXiv preprint*. arXiv: 1801.08287. 4, 37

Singh, S. P. and R. S. Sutton (1996). 'Reinforcement Learning with Replacing Eligibility Traces'. *Machine Learning* 22.1-3, pp. 123–158. 82

Sobel, M. (1982). 'The Variance of Discounted Markov Decision Processes'. *Journal of Applied Probability* 19.4, pp. 794–802. 32, 34, 35

Sutton, R. S. (1988). 'Learning to Predict by the Methods of Temporal Differences'. *Machine Learning* 3.1, pp. 9–44. 13

Sutton, R. S. (1989). *Implementation Details of the TD(λ) Procedure for the Case of Vector Predictions and Backpropagation*. Tech. rep. May 1987. GTE Laboratories Incorporated, pp. 1–4. 15

Sutton, R. S. and A. G. Barto (2018). *Reinforcement Learning: An Introduction*. 2nd ed. MIT Press. 10, 12, 15, 25, 58, 69

Sutton, R. S., H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári and E. Wiewiora (2009). 'Fast Gradient-Descent Methods for Temporal Difference Learning with Linear Function Approximation'. In: *Proceedings of the 26th Annual* 102

International Conference on Machine Learning - ICML '09. New York, New York, USA: ACM Press, pp. 1–8.

Sutton, R. S., A. R. Mahmood and M. White (2015). 'An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning'. *arXiv preprint*. arXiv: 1503.04269. 14, 16, 103

Sutton, R. S., J. Modayil, M. Delp, T. Degris, P. M. Pilarski and A. White (2011). 'Horde: A Scalable Real-Time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction'. In: *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*. AAMAS '11. Taipei: International Foundation for Autonomous Agents and Multiagent Systems, pp. 761–768. 7, 121

Sutton, R. S., C. Szepesvári and H. R. Maei (2009). 'A Convergent O(n) Temporal-difference Algorithm for Off-policy Learning with Linear Function Approximation'. *Advances in Neural Information Processing Systems*, pp. 1609–1616. 58

Tamar, A., D. D. Castro and S. Mannor (2016). 'Learning the Variance of the Reward-To-Go'. *Journal of Machine Learning Research* 17.13, pp. 1–36. 32, 35, 36, 38, 39, 68, 69, 71

Tsitsiklis, J. and B. Van Roy (1997). 'An Analysis of Temporal-Difference Learning with Function Approximation'. *IEEE Transactions on Automatic Control* 42.5, pp. 674–690. 14, 16, 19, 66

White, A. (2015). 'Developing a Predictive Approach to Knowledge'. PhD thesis. University of Alberta. 7, 121

White, M. and A. White (2016). 'A Greedy Approach to Adapting the Trace Parameter for Temporal Difference Learning'. In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. AAMAS '16. International Foundation for Autonomous Agents and Multiagent Systems, pp. 557–565. arXiv: 1607.00446. 32, 35–37, 39, 57

Williams, R. J. (1992). 'Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning'. *Machine Learning* 8.3-4, pp. 229–256. 6

Yu, H. (2017). 'On Convergence of some Gradient-based Temporal-Differences Algorithms for Off-policy Learning'. *arXiv preprint*. arXiv: arXiv:1712.09652v2. 102