Extending the Sliding-step Technique of Stochastic Gradient Descent to Temporal Difference Learning

by

Tian Tian

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

Master of Science

Department of Computing Science

University of Alberta

© Tian Tian, 2018

Abstract

Stochastic gradient descent is at the heart of many recent advances in machine learning. In each of a series of steps, stochastic gradient descent processes an example and adjusts the weight vector in the direction that would most reduce the error for that example. A step-size parameter is used to control the amount of adjustments. Altogether, the step-size parameter, the error and the direction of adjustment form the update to the weight vector.

Importance weighting is a common technique used in machine learning, where an example of a time step is weighted more than others. The scalars that weight the examples are called the importance weights, and they vary from time step to time step. Therefore, the update sizes fluctuate in proportion to the importance weights, which increases the chance for unstable behavior.

This thesis extends a technique that handles importance weights developed by Karampatziakis and Langford in 2011, which we refer to as the sliding-step technique. The outcome of the sliding-step technique is an expression that moderates the effect of the importance weights and choice of feature vectors on the size of the updates.

The primary contribution of this thesis is to extend sliding-step from the supervised learning setting to the temporal difference learning setting. For simplicity, we restrict our attention to the one step case, i.e. TD(0), with linear function approximation. We propose a new algorithm called sliding-step TD, and in the tabular case, we show it is convergent with probability one. Our empirical results suggest that sliding-step TD retains many of the favorable properties of the original supervised learning sliding-step algorithms. Finally, we consider applications to emphatic and residual-gradient algorithms, for which importance weightings are especially important.

Preface

No part of this thesis has been previously published.

To my mother

who made all this possible, and to the light of my life,

Ada

A journey of a thousand miles begins with a single step.

– Lao Zi, Chinese philosopher.

Acknowledgements

I thank foremost my supervisor, Dr. Richard S. Sutton for the contribution and support in completing this thesis. His insight and discussion on reinforcement learning have not only deepened my knowledge of the subject area but have also opened my mind to new possibilities. He has always encouraged me to look at things beyond the surface. The academic nurturing he has provided allowed me to grow as a researcher.

I would like to give special thanks to Dr. Huizhen Yu for sharing her vast knowledge on the theory of reinforcement learning. She has given me many of her thoughts on sliding-step TD, which has helped to organize my thoughts and complete this thesis.

I would also like to thank Roshan Shariff, our math guru for all the insightful discussions of math. I thank all the people of the Reading and Writing Group led by Martha Steenstrup for helping me with my writing and presentation. Lastly, I thank the rest of RLAI lab for all the intriguing and invigorating discussions from the topics of graphing to physics and to RL.

Lastly, I thank Chris and my parents for being my anchors through this self discovering journey.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Stochastic Gradient Descent

In recent years, deep learning has dominated the field of machine learning, making advances in natural language processing, image recognition, etc. At the heart of deep learning is the method stochastic gradient descent (SGD). SGD is especially useful in the on-line setting, where a stream of data is presented to a learning system one example at a time. Each example consists of an input $\boldsymbol{x} \in \mathbb{R}^n$, and a corresponding output $y \in \mathbb{R}$ of some unknown function of interest. The goal of the learning system is to find an approximation to this unknown function using only the observed input and output pairs. For each input, we can formulate an estimate that depends on the weight vector $\boldsymbol{w}$. An example of an estimate is a linear function of the input and the weight vector $\boldsymbol{x}^\top \boldsymbol{w}$, where $\top$ denotes the transpose. By adjusting $\boldsymbol{w}$ in the direction that would most reduce the difference between the estimate and the output of each example (i.e., error or loss), this changes how "close" or "far away" the approximation is to the actual function. Formally, SGD performs the following update to the weight vector,

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha \nabla loss(y_t, \boldsymbol{x}_t; \boldsymbol{w}_t),$$

where α is the step-size parameter that controls the amount of adjustment to $\boldsymbol{w}$ in each time step $t = 0, 1, 2, \cdots$. $\nabla loss(y_t, \boldsymbol{x}_t; \boldsymbol{w}_t)$ is the gradient of some loss function w.r.t. $\boldsymbol{w}$, and it defines the direction that would most reduce the error for the example $(\boldsymbol{x}_t, y_t)$. The stochasticity is due to the random choice

of the example.

1.2 Importance Weighting

Importance weighting is a common technique used in machine learning, where an example of a time step is weighted more than others. The scalars that weight the examples are called the importance weights, denoted by h throughout this thesis, and they vary from time step to time step. The update of SGD with an importance weight becomes $\alpha h \nabla loss$. If α is not made sufficiently small, a large h would certainly result in a large change to $\boldsymbol{w}$, which could increase the error instead of reducing it. This problem is illustrated in figure 1.1(a). Linearly scaling the gradient estimate increases the chance for unstable behavior, and we are less certain of the convergence of the algorithm. One way is to use a small enough step-size parameter that scales down the largest importance weight. However, this could mean unnecessarily small changes to $\boldsymbol{w}$ in other time steps and result in an overall slow down of learning.

1.3 The Sliding-step Technique in the Supervised Learning Setting

Karampatziakis and Langford (2011) introduce a new way of handling importance weights in the supervised learning setting, which we refer to as the sliding-step technique. The sliding-step technique can be described as follows: sub-divide the one update to the weight vector in one time step into k steps, each with a step-size parameter of $\frac{\alpha h}{k}$. In each of the k steps, we recompute the gradient estimate and adjust the weight vector in the newly computed direction. By tending $k \to \infty$ where the step-size parameter $\frac{ah}{k}$ becomes infinitesimal, we acquire a new class of algorithms. An illustration of this gradient procedure in one dimensional can be seen in figure 1.1(b), where the red point ($\boldsymbol{w}_t$, loss$(y_t, \boldsymbol{x}_t; \boldsymbol{w}_t)$) slides down the error surface towards the optimum that would achieve a zero loss for that example.

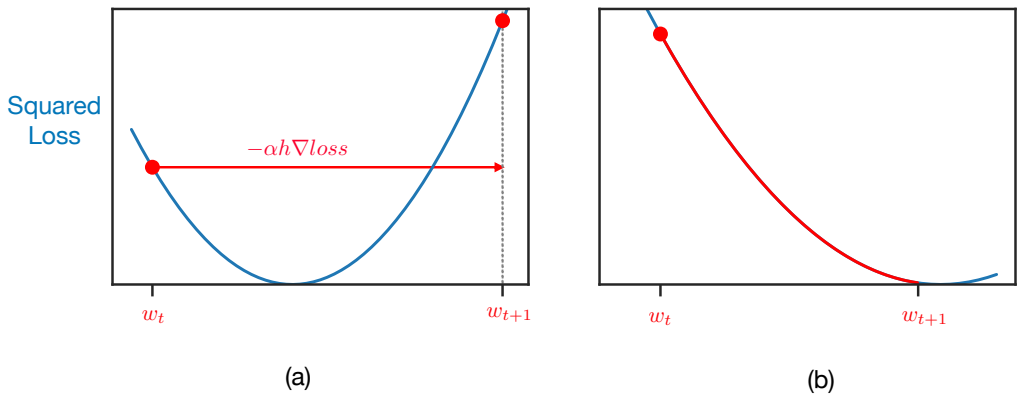

Figure 1.1: A demonstration of the sliding-step idea in one dimensional. (a) is the update made to $\boldsymbol{w}$ without the sliding-step idea (b) is the update made to $\boldsymbol{w}$ with the sliding-step idea. The blue curve is the error surface and in this case, it is the squared loss function $\frac{1}{2}(y_t - \boldsymbol{x}_t^\top \boldsymbol{w})^2$.

In the presence of a large importance weight, the sliding-step technique prevents the prediction from overshooting its target. This is because the gradient estimate is recomputed in each step, and while the prediction tends to y_t, the gradient estimate tends to 0. Thus, the error after the update with sliding-step is less than or equal to the error prior to the update.

The outcome of the sliding-step technique applied to the squared loss is the expression:

$$\frac{1 - \exp(-\alpha h \boldsymbol{x}^\top \boldsymbol{x})}{\boldsymbol{x}^\top \boldsymbol{x}} \tag{1.1}$$

replacing αh in the SGD update of $\alpha h \nabla loss$. Let us write $\boldsymbol{x}^\top \boldsymbol{x} = \|\boldsymbol{x}\|^2$, and the behavior of (1.1) as a function of αh is similar to:

$$\frac{\min(\alpha h \|\boldsymbol{x}\|^2, 1)}{\|\boldsymbol{x}\|^2}. \tag{1.2}$$

As matter of fact, (1.1) is upper bounded by (1.2) as illustrated in figure 1.2.

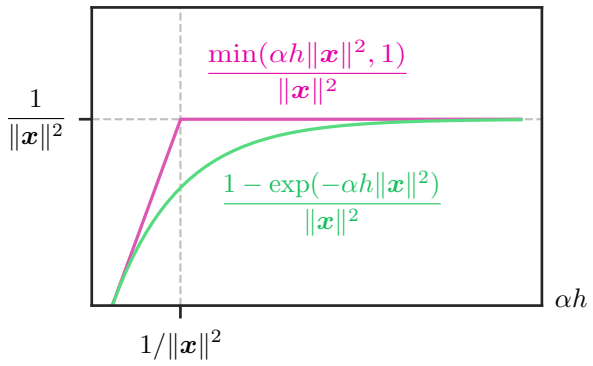

Figure 1.2: Comparing the sliding-step expression $\frac{1-\exp(-\alpha h\,\|\boldsymbol{x}\|^2)}{\|\boldsymbol{x}\|^2}$ to $\frac{\min(\alpha h\,\|\boldsymbol{x}\|^2,1)}{\|\boldsymbol{x}\|^2}$

When $\alpha h \geq \frac{1}{\|\boldsymbol{x}\|^2}$, (1.2) truncates αh to $\frac{1}{\|\boldsymbol{x}\|^2}$. Similarly, the expression (1.1) also truncates αh to $\frac{1}{\|\boldsymbol{x}\|^2}$ when αh is large but not at $\alpha h = \frac{1}{\|\boldsymbol{x}\|^2}$ exactly. Nonetheless, the effect of the truncation bounds the size of the update to the weight vector, and hence reduces the variance of the iterates at the expense of bias. When $\alpha h < \frac{1}{\|\boldsymbol{x}\|^2}$, (1.2) is αh, which deduces to the original SGD update of $\alpha h \nabla loss$. In this case, (1.1) will be approximately αh and the bias introduced will be small. $\frac{1}{\|\boldsymbol{x}\|^2}$ appears in (1.1) also normalizes the input $\boldsymbol{x}$, which again bounds the size of the update.

1.4 The Contribution of this Thesis

In this thesis, we extend the sliding-step technique from supervised learning to temporal difference learning. In supervised learning, the data are independently identically distributed (i.i.d.), where each example can be processed in any order while in temporal difference learning the data are sequential. Because of the sequential nature of the data, TD algorithms use bootstrapping, where the target is also an estimate, or some parameterized function of the weight vector. This leads to many choices that can be made in terms of the extension. Due to these considerations, the extension we discuss in this thesis is not trivial. We restrict our attention to the one-step case with linear function approximation, and thus we omit writing the usual (0) as in TD(0). We propose a new algorithm called sliding-step TD, and in the tabular case

with on-policy training, we show it to be convergent with probability one. Our empirical results suggest that sliding-step TD retains many of the favorable properties of the original supervised learning sliding-step algorithms. Finally, we consider applications to emphatic TD and residual-gradient TD, for which importance weightings are especially important.

Chapter 2

Background

This chapter defines terminologies and gives background information necessary for this thesis. Readers who are familiar with these subjects can skip to chapter 3 without any loss of continuity.

In section 2.1, we will use $n \in \mathbb{N}$ to denote the discrete time steps instead of t and let $t \in \mathbb{R}$. The reason for the change of notation is to make the explanation of section 2.1 more clear. However, for the rest of this chapter and thesis, we go back to using t to denote the discrete time steps, so to be aligned with the notation in Sutton and Barto (2018).

2.1 Stochastic Gradient Descent

In supervised learning, we have a series of examples $(\boldsymbol{x}_n, y_n)_{n\geq 0}$, where $\boldsymbol{x}_n \in \mathbb{R}^m$ is the input and y_n is its target. In general, y_n can be a vector of real numbers, but for simplicity sake, we denote $y_n \in \mathbb{R}$. The set $\{(\boldsymbol{x}_n, y_n)\}$ is assumed to be i.i.d. The goal is to find a weight vector $\boldsymbol{w} \in \mathbb{R}^m$ that minimizes the expected loss between the target and its estimate $\hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_n)$, where $\hat{y}_{\boldsymbol{w}}$ belongs to a family of parameterized functions $\{\hat{y}_{\boldsymbol{w}} : \mathbb{R}^m \to \mathbb{R}\}$. For an example, we can consider the squared loss $\frac{1}{2}\big(y_n - \hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n)\big)^2$, where the expected loss is the mean squared error given by $f(\boldsymbol{w}) \doteq \frac{1}{2}\mathbb{E}\big[y_n - \hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n)\big]^2$. Then, the goal is to find:

$$\boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}} f(\boldsymbol{w}).$$

The problem is that the expectation in $f(\boldsymbol{w})$ cannot be evaluated because the underlying distribution in which the examples are drawn from is unknown. In addition, if only an example $(\boldsymbol{x}_n, y_n)$ is made available at a time, then we can use stochastic gradient descent (SGD) to obtain $\boldsymbol{w}^\star$ in an iterative fashion. In each step, SGD adjusts the weight vector $\boldsymbol{w}$ by a small amount in the direction that would reduce the loss the most for that example:

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \alpha_n\big(y_n - \hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n)\big)\nabla\hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n). \tag{2.1}$$

The direction that reduces the loss the most is the direction negative of the gradient of the squared loss, which is $-\big(y_n - \hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n)\big)\nabla\hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n)$. The amount of adjustment is controlled by a sequence of step-size parameters $(\alpha_n)_{n\geq 0}$ satisfying the step-size conditions (Robbins and Monro, 1951) while retaining $\alpha_n \to 0$, where upon convergence, SGD obtains $\boldsymbol{w}^\star$. In the next subsection, we introduce the step-size conditions and motivate why we need them.

2.1.1 The Step-size Parameters

For a sequence $(\alpha_n)_{n\geq 0}$, the step-size conditions (Robbins and Monro, 1951) state:

$$\sum_{n=0}^{\infty} \alpha_n = \infty \tag{2.2}$$

$$\sum_{n=0}^{\infty} \alpha_n^2 < \infty. \tag{2.3}$$

To motivate why the step-size parameters must satisfy these conditions, we follow similar presentation of Borkar (2008). First, we rewrite (2.1) as follows,

$$\begin{aligned}
\boldsymbol{w}_{n+1} &= \boldsymbol{w}_n + \alpha_n\big(y_n - \hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n)\big)\nabla\hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n) + \alpha_n\nabla f(\boldsymbol{w}_n) - \alpha_n\nabla f(\boldsymbol{w}_n) \\
&= \boldsymbol{w}_n + \alpha_n\big(-\nabla f(\boldsymbol{w}_n)\big) \\
&\quad + \alpha_n\Big(\big(y_n - \hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n)\big)\nabla\hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n) - \mathbb{E}\big[\big(y_n - \hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n)\big)\nabla\hat{y}_{\boldsymbol{w}_n}(\boldsymbol{x}_n)\big|\boldsymbol{w}_n\big]\Big) \\
&= \boldsymbol{w}_n + \alpha_n\big(-\nabla f(\boldsymbol{w}_n) + M_{n+1}\big),
\end{aligned} \tag{2.4}$$

where $(M_n)_{n\geq 0}$ is regarded as a sequence of noise. The reason why (M_n) can be regarded as a sequence of noise is because it satisfies $\mathbb{E}[M_n|\boldsymbol{w}_m, m \leq n] = 0$

for $n \geq 0$ [1]. In other words, it is a sequence of zero mean random variables uncorrelated with the past. Therefore, we can view (2.4) as a noisy discretized version of the following ordinary differential equation (o.d.e),

$$\dot{\boldsymbol{w}}(t) = -\nabla f\big(\boldsymbol{w}(t)\big),$$

which is known to converge to a local minimum of f. Since we are interested in the asymptotic behavior of this o.d.e., we must ensure that $(\alpha_n)_{n\geq 0}$ cover the entire time axis t. In other words, $(\alpha_n)_{n\geq 0}$ must satisfy (2.2) while retaining $\alpha_n \to 0$ so that in the limit, the asymptotic behavior of (2.1) tracks that of the o.d.e. The second property (2.3) ensures both the discretization error and the noise goes to 0 asymptotically with probability 1. In addition, the step-size conditions (2.2) and (2.3) give guidelines to how fast the sequence α_n should decay. The step-size parameters of SGD are crucial to its performance and convergence.

In practice, we often resort to a constant step-size parameter α instead. A constant step-size parameter is especially useful in the scenario where the environment is changing slowly, slower than the algorithm can be made to adapt. In this case, $\alpha > 0$ allows the algorithm to keep making changes to its weight vector, and the learning continues. The analysis of SGD with a constant step-size parameter can be found in Borkar (2008) and Kushner and Yin (2003). With a constant step-size parameter, we can no longer talk about probability 1 convergence to the solution $\boldsymbol{w}^\star$ but that the iterates will concentrate within a neighborhood of $\boldsymbol{w}^\star$ with high probability. The neighborhood can be made small by smaller α, but smaller α results in smaller adjustments to the weight vector, and thus slower learning. While a bigger α results in larger adjustments to the weight vector, but risks instability and divergence in the long run. The size of α trades off the rate of learning and how close is to $\boldsymbol{w}^\star$.

[1] Any sequence that satisfies $\mathbb{E}[M_n|\boldsymbol{w}_m, m \leq n] = 0$ for $n \geq 0$ is called a martingale difference sequence

2.1.2 A Few Common Step-size Algorithms

Manually setting a step-size parameter is often difficult because the characteristic of the dataset is unknown and searching for it may not be easily afforded. There has been many step-size tuning algorithms for SGD, and to name a few, ADAGRAD (Duchi et al., 2011), ADAM (Kingma and Ba, 2015) and RMSProp (Hinton et al., 2012) are amongst the most common. ADAGRAD tune the step-size parameter for each component of the weight vector. ADAGRAD keeps a running sum of the squares of the gradients with respect to each component of the weight vector up to step t. The step-size parameter for that component is one over the square root of this value plus some epsilon to keep the algorithm from division by zero. Constructing the step-size parameter this way has its drawbacks. The step-size parameter is one over the sum, which can become extremely small when the sum is large. The problem is that after some iterations, the changes made to the weight vector become minuscule, and the iterates stop short of the local optimum. RMSProp addresses this problem by using an exponentially decayed average of the squared gradients instead. Thus, the sum is bounded, and one over the square root of the sum prevents the step-size from becoming too small. ADAM is another step-size tuning algorithm. It keeps a decaying sum of the gradients and a decaying sum of the gradients squared. The former estimates the mean of the gradients while the latter estimates the second moment of the gradients. The step-size parameter becomes one over the square root of the second moment and the mean estimate replaces the gradient term.

ADAGRAD, RMSProp and ADAM are just a few step-size algorithms. Although the sliding-step algorithms also belong to this vast class of step-size algorithms, but we will not compare ADAGRAD, RMSProp and ADAM with the sliding-step algorithms. In this thesis, we limit our focus on comparing the sliding-step algorithms with their respective standard counterparts.

2.2 Temporal Difference (TD) Learning

Temporal difference (TD) learning is about making long term predictions by exploiting the temporal structure of the data. Before we define what long term predictions are, we consider the framework in which TD is build upon. We consider an infinite-horizon Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, \mathcal{R}, \gamma)$ with finite state set $\mathcal{S}$, action set $\mathcal{A}$, and reward set $\mathcal{R}$. The discount factor is $\gamma \in [0, 1]$. We use $|\mathcal{S}|$ to denote the cardinality of the state space $\mathcal{S}$.

At each discrete time step t, an agent in state $S_t \in \mathcal{S}$ takes an action $A_t \in \mathcal{A}$ according to a given fixed policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ where $\sum_a \pi(s, a) = 1$. In combination with the dynamics of the environment determined by the state-transition probability $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$:

$$\begin{aligned} p(s'|s, a) &\doteq Pr\{S_{t+1} = s'|S_t = s, A_t = a\} \\ &= \sum_{r \in \mathcal{R}} p(s', r|s, a), \end{aligned}$$

the agent transitions from the current state S_t to the next state $S_{t+1} \in \mathcal{S}$, obtaining a numerical reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$. We shall use $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ to denote the expected rewards for a state and action pair:

$$\begin{aligned} r(s, a) &\doteq \mathbb{E}_\pi \big[R_{t+1} \big| S_t = s, A_t = a\big] \\ &= \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a). \end{aligned} \tag{2.5}$$

A sample of MDP comes in the form of an episode, which is comprised of a sequences of states, actions and rewards, $(S_0, A_0, R_1, S_1, A_1, R_2, \cdots, S_T)$, where S_T denotes either a terminal state or a state up to some horizon. In this thesis, we focus on the evaluation of a policy rather than decision making. The policy is known and fixed throughout all time step t. Even though the action selections based on the policy will affect the process of state to state transitions, $(S_0, R_1, S_1, R_2, \cdots, S_T)$ will suffice in the computation of the state values. Thus, an example of a sample (S_t, R_{t+1}, S_{t+1}) consists of the current state, the reward, and the next state.

2.2.1 Policy Evaluation

The idea of a long term prediction is captured by the return

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots ,$$

which is a sum of discounted rewards from state S_t. To evaluate a policy π, we mean to evaluate the following quantity for every state $s \in \mathcal{S}$,

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\big[G_t \big| S_t = s\big] \\ &= \mathbb{E}_\pi\big[R_{t+1} + \gamma G_{t+1} \big| S_t = s\big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big(r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\Big) \\ &= \sum_a \pi(a|s)\Big(r(s, a) + \gamma \sum_{s'} p(s'|s, a) v_\pi(s')\Big). \end{aligned} \tag{2.6}$$

Equation (2.6) is the Bellman equation and it holds true for any finite MDP $\mathcal{M}$ and policy π.

We use $\boldsymbol{v}_\pi \in \mathbb{R}^{|\mathcal{S}|}$ to denote a vector of state values where each component of this vector is $v_\pi(s)$. Likewise, $\boldsymbol{r}_\pi \in \mathbb{R}^{|\mathcal{S}|}$, where the sth component $[\boldsymbol{r}_\pi]_s \doteq \mathbb{E}_\pi[R_{t+1}|S_t = s]$, which is (2.5) with actions marginalized out. Now we can rewrite (2.6) using the Bellman operator $T^\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$:

$$(T^\pi \boldsymbol{v}_\pi) \doteq \boldsymbol{r}_\pi + \gamma \mathbf{P}_\pi \boldsymbol{v}_\pi = \boldsymbol{v}_\pi \quad \text{i.e.,} \quad \boldsymbol{v}_\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \boldsymbol{r}_\pi, \tag{2.7}$$

where $\mathbf{P}_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ is the state transition matrix of the induced Markov chain by the policy π, has its i, jth component $[\mathbf{P}_\pi]_{ij} = \sum_{a\in\mathcal{A}} p(j|i, a)\pi(a|i)$.

2.2.2 On-policy and Off-policy Training

There are two common ways to evaluate the policy π, by ways of on-policy and off-policy training. In on-policy training, state values of policy π is evaluated using data generated by the same policy. In off-policy training, the state values of policy π is evaluated using data generated by another policy, the behavior policy μ.

In off-policy training, the behavior policy μ selects the agent's action. As a result, the state visitation and the distribution over the observed rewards

will be different from following policy π. Using importance sampling, we can obtain an unbiased estimate of the return under π from state s (i.e., $v_\pi(s)$). Consider the following return where the rewards are generated via policy μ,

$$G_t^\rho = \rho_t R_{t+1} + \gamma \rho_t \rho_{t+1} R_{t+2} + \gamma^2 \rho_t \rho_{t+1} \rho_{t+2} R_{t+3} + \cdots, \tag{2.8}$$

where

$$\rho_t \doteq \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$$

is called the importance sampling ratio, assuming $\mu(a|s) > 0$ for every state and action for which $\pi(a|s) > 0$. We can rewrite (2.8) in the following recursive form,

$$G_t^\rho = \rho_t (R_{t+1} + \gamma G_{t+1}^\rho)$$

and show that $\mathbb{E}_\mu\big[G_t^\rho \big| S_t = s\big] = \mathbb{E}_\pi\big[G_t \big| S_t = s\big] = v_\pi(s)$.

Proof.

$$\begin{aligned}
\mathbb{E}_\mu\big[G_t^\rho \big| S_t = s\big] &= \mathbb{E}_\mu\left[\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}\Big(R_{t+1} + \gamma G_{t+1}^\rho\Big)\,\Big|\, S_t = s\right] \\
&= \sum_a \mu(a|s)\frac{\pi(a|s)}{\mu(a|s)}\Big(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\mathbb{E}_\mu\big[G_{t+1}^\rho \big| S_{t+1} = s'\big]\Big) \\
&= \mathbb{E}_\pi\Big[R_{t+1} + \gamma \mathbb{E}_\mu\big[G_{t+1}^\rho \big| S_{t+1} = s'\big]\,\Big|\, S_t = s\Big],
\end{aligned}$$

which continues to roll out, and we get unbiased estimate of the state values according to π. □

If the two policies are the same (i.e. $\mu = \pi$ where $\rho = 1$ for all state and action pairs), then we return to the on-policy case.

2.2.3 Linear Function Approximation

We consider a class of linear functions that are parameterized by $\boldsymbol{w}$ to approximate v_π:

$$v_\pi(s) \approx \boldsymbol{x}(s)^\top \boldsymbol{w} \doteq v(s, \boldsymbol{w}),$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is called the weight vector and $\boldsymbol{x}(s) \in \mathbb{R}^n$ is called the feature vector of state s. Each component of $\boldsymbol{x}(s)$ is a feature of the state. The feature

vectors are numerical representations of the underlying states. Throughout this thesis, we let $\boldsymbol{x}_t = \boldsymbol{x}(S_t)$, and both notations will be used interchangeably.

The feature matrix is formed by gathering all the feature vectors and stack them into a matrix:

$$\boldsymbol{X}_{|\mathcal{S}|\times n} = \begin{bmatrix} - & \boldsymbol{x}(s_1)^\top & - \\ - & \boldsymbol{x}(s_2)^\top & - \\ & \cdots & \\ - & \boldsymbol{x}_(s_{|\mathcal{S}|})^\top & - \end{bmatrix}.$$

The columns of $\boldsymbol{X}$ spans an approximation subspace of dimension n where the linear estimator $\boldsymbol{X}\boldsymbol{w}$ resides. If $n = |\mathcal{S}|$ and full rank, then the state value function $\boldsymbol{v}_\pi$ for every state can be represented exactly. A special case of $\boldsymbol{X}$ is the identity matrix with dimension $|\mathcal{S}| \times |\mathcal{S}|$, and this is the function approximation equivalent of the tabular case, where we learn each individual state values independently and asynchronously. If $n < |\mathcal{S}|$, then not all of the state value functions will be represented exactly, and the goal is to find $\boldsymbol{w}^\star$ such that $\Pi(T^\pi v(s, \boldsymbol{w}^\star)) = v(s, \boldsymbol{w}^\star)$ for all $s \in \mathcal{S}$, a projected version of the Bellman equation (2.7).

2.2.4 TD Learning Algorithms

In the n-step learning case, the target of learning consists of $R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n}$ plus the bootstrapped value of the state S_{t+n}. In this thesis, we consider the one-step learning case. We will apply the Sliding-step idea to the following TD learning algorithms:

TD: (Sutton and Barto, 1998, 2018)
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \delta_t \boldsymbol{x}_t \tag{2.9}$$

Emphatic TD: (Sutton et al., 2016)
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha F_t \delta_t \boldsymbol{x}_t \tag{2.10}$$
$$F_t = \gamma F_{t-1} + i(S_t) \text{ and } F_0 = 1$$

Residual-gradient TD: (Baird, 1995)
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \delta_t (\boldsymbol{x}_t - \gamma \boldsymbol{x}'_{t+1}) \tag{2.11}$$

The TD error $\delta_t = R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t$ is comprised of the target $R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t$ and the prediction $\boldsymbol{x}_t^\top \boldsymbol{w}_t$. The target is comprised of the reward and a bootstrapped value of the next state, where an estimate of the

true value function is used instead. The $\boldsymbol{x}'_{t+1}$ is the feature vector of another independently sampled next state.

The emphasis $F \in [0, \infty)$ of Emphatic TD emphasizes the update of a state s based on how much the state is bootstrapped in addition to the interest $i : \mathcal{S} \to [0, \infty)$ expressed for that state. Interests weigh certain states more than others, which shift the accuracy of the estimator toward those states. Prior to interest, the accuracy of the estimator depends only on the state visitation; more visited states will have more accurate estimates than the less visited states. However, the state visitation is part of the process itself and difficult to manipulate. The interest function circumvents this and provides added flexibility where it can be used to re-weight the states according to an user's interest. Emphatic TD takes interest into account and is a stable algorithm under off-policy training (Sutton et al., 2016; Yu, 2015).

The gradient estimate $\delta_t \boldsymbol{x}_t$ is a "partial" gradient in that it only considers the effect of changing $\boldsymbol{w}_t$ on the prediction (i.e., $\boldsymbol{x}_t^\top \boldsymbol{w}_t$) while ignoring the effect on the target $R_{t+1} + \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t$. These gradient estimates are called semi-gradients by Sutton and Barto (2018), and $\delta_t \boldsymbol{x}_t$ is not the gradient of any objective function. TD (2.9) and emphatic TD (2.10) are semi-gradient algorithms, while residual-gradient TD (2.11) is a gradient algorithm.

Residual-gradient TD performs gradient descent on the Mean Squared Bellman Error (MSBE):

$$\begin{aligned} MSBE(\boldsymbol{w}) &\doteq \sum_{s \in \mathcal{S}} d_\pi(s) \Big(T^\pi v(s, \boldsymbol{w}) - v(s, \boldsymbol{w}) \Big)^2 \\ &= \sum_{s \in \mathcal{S}} d_\pi(s) \Big(\mathbb{E}_\pi \big[R_{t+1} + \gamma v(S_{t+1}, \boldsymbol{w}) \big| S_t = s \big] - v(s, \boldsymbol{w}) \Big)^2, \quad (2.12) \end{aligned}$$

where d_π is the fraction of time spent in state s. For a continuing task, d_π is the stationary distribution. If the policy and/or the environment is non-deterministic, residual-gradient TD requires two independent samples of the next state S_{t+1}^1 and S_{t+1}^2 (i.e., double sampling) in order to converge to the minimum of MSBE. Otherwise, residual-gradient TD converges to the minimum of Mean Squared TD Error (i.e., $\mathbb{E}[\delta_t^2]$), which in general does not satisfy the Bellman equation (2.7).

In the limit, algorithms (2.9) to (2.10) aim to solve a projected version of the Bellman equation (2.7):

$$\begin{aligned} V_{\boldsymbol{w}^\star}(s) &= \Pi(\boldsymbol{r}_\pi + \gamma \mathbf{P}_\pi V_{\boldsymbol{w}^\star}(s)) \\ &= \Pi(T^\pi V_{\boldsymbol{w}^\star}(s)) \end{aligned}$$

for all $s \in \mathcal{S}$. Each differ in the weighting on the projection Π. For an example of a projection Π, see the next section.

2.2.5 Fixed Point of On-policy TD

Linear TD is defined by the following stochastic algorithm,

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \alpha\Big(\underbrace{R_{t+1}\boldsymbol{x}_t}_{\mathbf{b}_t \in \mathbb{R}^n} - \underbrace{\boldsymbol{x}_t(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1})^\top}_{\mathbf{A}_t \in \mathbb{R}^{n\times n}} \boldsymbol{w}_t \Big) \\ &= \boldsymbol{w}_t + \alpha(\mathbf{b}_t - \mathbf{A}_t \boldsymbol{w}_t), \end{aligned} \tag{2.13}$$

which in an average sense behaves like the following deterministic algorithm,

$$\bar{\boldsymbol{w}}_{t+1} \doteq \bar{\boldsymbol{w}}_t + \alpha(\mathbf{b} - \mathbf{A}\bar{\boldsymbol{w}}_t) \tag{2.14}$$

when the process $(S_t, S_{t+1}, \boldsymbol{x}(S_t))$ reaches the steady state. The matrix:

$$\begin{aligned} \mathbf{A} &= \sum_s \boldsymbol{d}_\pi(s) \mathbb{E}_{\boldsymbol{d}_\pi}\big[\boldsymbol{x}(S_t)(\boldsymbol{x}(S_t) - \gamma \boldsymbol{x}(S_{t+1}))^\top \big| S_t = s\big] \\ &= \sum_s \boldsymbol{x}(s)\big(\boldsymbol{x}(s) - \sum_{s'} \gamma [\mathbf{P}_\pi]_{ss'} \boldsymbol{x}(s')\big) \\ &= \boldsymbol{X}^\top \mathbf{D}_\pi (\mathbf{I} - \gamma \mathbf{P}_\pi)\boldsymbol{X}, \end{aligned}$$

and the vector:

$$\mathbf{b} = \boldsymbol{X}^\top \mathbf{D}_\pi \boldsymbol{r}_\pi,$$

where $\mathbf{D}_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ is a diagonal matrix of the steady state distribution $\boldsymbol{d}_\pi$. By Sutton (1988) and Tsitsiklis and Van Roy (1997), $\mathbf{A}$ has a full set of eigenvalues all of whose real parts are positive. Thus, we can invert $\mathbf{A}$ and obtain $\bar{\boldsymbol{w}}$ by solving $\mathbf{A}\boldsymbol{w} = \mathbf{b}$. We say that the stochastic algorithm (2.13) is stable if the deterministic algorithm (2.14) converges to the unique fixed

point $\bar{\boldsymbol{w}}$ independent of the initial $\bar{\boldsymbol{w}}_0$. By Tsitsiklis and Van Roy (1997), the stochastic algorithm (2.13) do indeed converge to the fixed point $\bar{\boldsymbol{w}} = \mathbf{A}^{-1}\mathbf{b}$.

By solving $\mathbf{A}\boldsymbol{w} = \mathbf{b}$, the exact form of $\bar{\boldsymbol{w}}$ using the Bellman operator is as follows,

$$\bar{\boldsymbol{w}} = (\boldsymbol{X}^\top \mathbf{D}_\pi \boldsymbol{X})^{-1} \boldsymbol{X}^\top \mathbf{D}_\pi (T^\pi \boldsymbol{X} \bar{\boldsymbol{w}}). \tag{2.15}$$

Then, it follows

$$\boldsymbol{X}\bar{\boldsymbol{w}} = \Pi(T^\pi \boldsymbol{X}\bar{\boldsymbol{w}}),$$

where $\Pi = \boldsymbol{X}(\boldsymbol{X}^\top \mathbf{D}_\pi \boldsymbol{X})^{-1}\boldsymbol{X}^\top \mathbf{D}_\pi$ is the projection matrix weighted by $\mathbf{D}_\pi$ (Tsitsiklis and Van Roy, 1997). This is why we say that TD is solving a projected version of the Bellman equation (2.7).

2.2.6 Evaluations of TD Learning Algorithms

The algorithms examined in this thesis learn in an on-line and iterative fashion, and each algorithm produces a sequence of iterates $(\boldsymbol{w}_t)_{t\geq 0}$. We study the performance of the algorithms by evaluating their iterates against some performance measure. The iterate can be evaluated after every step or after each episode has ended. Although there are many evaluation methods in machine learning, we consider two of the most common methods used in reinforcement learning: the learning curve and the sensitivity graph. We can gather all the evaluations and plot the performance measure against steps or episodes as in a learning curve graph, or we can average all the performance measures for all steps or episodes and plot the summary against a set of chosen algorithmic parameter value as in a sensitivity graph.

A learning curve reveals two aspects of the algorithm: 1) how fast the algorithm improves performance and 2) what is the asymptotic performance of the algorithm. If the performance measure is some error (i.e., distance metric between the true value and the estimate), then the beginning of the learning curve reveals how quickly this error goes to zero, and the tail of the learning curve reveals how close is the error to zero.

A sensitivity graph reveals the robustness of an algorithm, meaning how sensitive an algorithm is toward changes in an algorithmic parameter. For a specific domain, an algorithm may be stable within some range of an algorithmic parameter and sensitivity graph may also show what that range is.

One repeat of an experiment may generate data that favor one algorithm over another, where in fact the one that seems to perform better is not better on average. Therefore each experiment is repeated n number of times called the runs, and each run is independent of the others. For every evaluation of an algorithm (by steps or episodes), there will be n i.i.d performance measures $X_1, \ldots X_n$, and this counts as one sample. We can then compute the sample mean ($\bar{X}$) and sample variance (s^2).

When comparing two sample means, it is common practice to examine the confidence interval or standard error. The confidence interval is defined as

$$\bar{X} \pm t^* \sqrt{\frac{s^2}{n}},$$

where t^* is the critical value of a t-distribution. We can see that the standard error is the confidence interval with the critical value equals 1, which corresponds to roughly 95% confidence level. If the confidence interval of the two means do not overlap, then the two means are statically different. If the confidence interval does overlap, it is not necessarily true that they are not statistically different. When in this situation, we can do more runs. The variance of the sample mean is $\frac{\sigma^2}{n}$, which diminishes at a rate of $\frac{1}{n}$. Smaller the variance of the sample mean implies smaller confidence interval, which leads to more statistical power.

To generate a learning curve for an algorithm, there will be a sample of performance measures, one sample for every step or episode. It is the sample means that are plotted against the steps or episodes. To generate a sensitivity graph for an algorithm, there will be a sample of sums for every value of the algorithmic parameter. It is the mean of the sums that are plotted against the values of the algorithmic parameter. In this thesis, we use standard error $\frac{s}{\sqrt{n}}$ of the sample mean as error bars in each plot.

Because we are interested in the quality of the solution, and by which, we

mean how closely the estimate of the state values approximate the true value of the states weighted by their respective state visitation. Then, Root Mean Squared Value Error (RMSVE):

$$\text{RMSVE}(\boldsymbol{w}) \doteq \sqrt{\sum_{s \in \mathcal{S}} \boldsymbol{d}_\mu(s)(v_\pi(s) - v(\boldsymbol{w}, s))^2} \tag{2.16}$$

seems to be most appropriate in evaluating the performance of our algorithms. $v_\pi(s)$ is the value of the state s if it were sampled according to policy π and $v(\boldsymbol{w}, s) = \boldsymbol{w}^\top \boldsymbol{x}(s)$ is the linear function approximation of $v_\pi(s)$. The RMSVE measures how well the linear estimator $\boldsymbol{w}^\top \boldsymbol{x}(\cdot)$ estimates all of the state values weighted by how often the state s is visited. How often a state is visited is according to the steady-state distribution of the Markov chain induced by $\boldsymbol{d}_\mu$, the behavior policy. If the algorithm is on-policy, then $\boldsymbol{d}_\mu$ is $\boldsymbol{d}_\pi$ for all state-action pairs. For some problems, we compute RMSVE after every step and for others, we compute RMSVE after every episode.

2.3 Importance Weighting in Machine Learning

Importance weighting is a common technique that appears in statistics and in many fields of machine learning. Importance weights are non-negative values that quantify the relative importance of one example over another. For an example, active learning (Beygelzimer et al., 2009) weight unlabeled examples by importance weights to remove sampling bias, and thus it converges to the optimal function in the class. Another example is boosting (Freund and Schapire, 1995), where the importance weights reflect the accuracy of the classification in previous iterations, and the accuracy of the weak hypothesis (or strategies) are adapted rather than specified a priori.

In TD learning, the importance sampling ratio described in section 2.2.2 and emphasis described in section 2.2.4 are analogous to the importance weights of the supervised learning, but differ in their purpose and how they transpire. We will first discuss importance sampling ratio and then emphasis.

In the one-step off-policy TD learning, the update to the weight vector is

weighted by an importance sampling ratio. The importance sampling ratio at time step t depends only on the state encountered and the action selected at t. For the same state and action observed in different time steps, the importance sampling ratio is the same.

The emphasis of emphatic TD is a sum of discounted interest of all the previously visited states, and at time step t, it can be express as follows,

$$F_t = \gamma^t(\rho_0 \cdots \rho_{t-1}) + \sum_{k=1}^{t} \gamma^{t-k} i(S_k)(\rho_k \cdots \rho_{t-1}).$$

Emphasis depends on the sequence of states $(S_0, S_1, \cdots, S_t)$ leading up to state S_t. If the same state is visited in different time steps, the emphasis weighting is different.

Chapter 3

Extending the Sliding-step Technique to TD Learning

This chapter concerns the first main contribution of the thesis. We talk about how we extend the sliding-step technique to temporal difference (TD) learning, and then propose a new algorithm called sliding-step TD.

For the rest of this thesis, we consider only the squared loss $\frac{1}{2}(y - \boldsymbol{x}^\top \boldsymbol{w})^2$.

3.1 The Karampatziakis and Langford's Sliding-step Technique

In supervised learning, it is common to use importance weights to quantify the relative importance of one example $(\boldsymbol{x}_t, y_t)$ over another. One way of handling importance weight in stochastic gradient descent is to multiply h_t with the gradient of the loss:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha h_t (y_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t) \boldsymbol{x}_t. \tag{3.1}$$

In the presence of a large importance weight, the estimate $\boldsymbol{x}_t^\top \boldsymbol{w}_{t+1}$ can overshoot y_t and achieve a large loss for that example (see figure 1.1(a)). We could make the alpha small enough for the largest h, but then this could mean unnecessarily small adjustments in the other updates. Small alpha slows down overall learning.

Let us consider an alternative way of handling importance weight. At time step t, we acquire an example $(\boldsymbol{x}, y)$ with importance weight h. Instead of

assigning the update $\boldsymbol{w}_t + \alpha h(y - \boldsymbol{x}^\top \boldsymbol{w}_t)\boldsymbol{x}$ to $\boldsymbol{w}_{t+1}$, we sub-divide it into k steps. In the first step, we compute the gradient of the loss at $\boldsymbol{w}_t$, and make the first intermediate update to the weight vector with an adjustment of $\frac{\alpha h}{k}$:

$$\tilde{\boldsymbol{w}}_1 = \boldsymbol{w}_t + \frac{\alpha h}{k}(y - \boldsymbol{x}^\top \boldsymbol{w}_t)\boldsymbol{x}. \tag{3.2}$$

In the second step, we recompute the gradient of the loss at $\tilde{\boldsymbol{w}}_1$ (i.e., $-(y - \boldsymbol{x}^\top \tilde{\boldsymbol{w}}_1)\boldsymbol{x}$), and make a second intermediate update to the weight vector along this newly computed direction with an adjustment of $\frac{\alpha h}{k}$:

$$\tilde{\boldsymbol{w}}_2 = \tilde{\boldsymbol{w}}_1 + \frac{\alpha h}{k}(y - \boldsymbol{x}^\top \tilde{\boldsymbol{w}}_1)\boldsymbol{x}. \tag{3.3}$$

By swapping the expression (3.2) into $\tilde{\boldsymbol{w}}_1$ of (3.3) and expand the recursion:

$$\begin{aligned}\tilde{\boldsymbol{w}}_2 &= \boldsymbol{w}_t + \frac{\alpha h}{k}(y - \boldsymbol{x}^\top \boldsymbol{w}_t)\boldsymbol{x} + \frac{\alpha h}{k}\left(y - \boldsymbol{x}^\top \left(\boldsymbol{w}_t + \frac{\alpha h}{k}(y - \boldsymbol{x}^\top \boldsymbol{w}_t)\boldsymbol{x}\right)\right)\boldsymbol{x} \\ &= \boldsymbol{w}_t + \left(\frac{2\alpha h}{k} - \left(\frac{\alpha h}{k}\right)^2 \boldsymbol{x}^\top \boldsymbol{x}\right)(y - \boldsymbol{x}^\top \boldsymbol{w}_t)\boldsymbol{x}.\end{aligned}$$

After the third step, we obtain:

$$\tilde{\boldsymbol{w}}_3 = \boldsymbol{w}_t + \left(3\left(\frac{\alpha}{k}\boldsymbol{x}_t^\top \boldsymbol{x}_t\right) - 3\left(\frac{\alpha}{k}\boldsymbol{x}_t^\top \boldsymbol{x}_t\right)^2 + \left(\frac{\alpha}{k}\boldsymbol{x}_t^\top \boldsymbol{x}_t\right)^3\right)(y - \boldsymbol{x}^\top \boldsymbol{w}_t)\boldsymbol{x}.$$

After performing k intermediate weight updates, we obtain:

$$\tilde{\boldsymbol{w}}_k = \boldsymbol{w}_t + \frac{1 - \left(1 - \frac{\alpha h}{k}\boldsymbol{x}^\top \boldsymbol{x}\right)^k}{\boldsymbol{x}^\top \boldsymbol{x}}(y - \boldsymbol{x}^\top \boldsymbol{w}_t)\boldsymbol{x}, \tag{3.4}$$

and we set $\boldsymbol{w}_{t+1} = \tilde{\boldsymbol{w}}_k$. This procedure visualized in four steps is illustrated in figure 3.1.

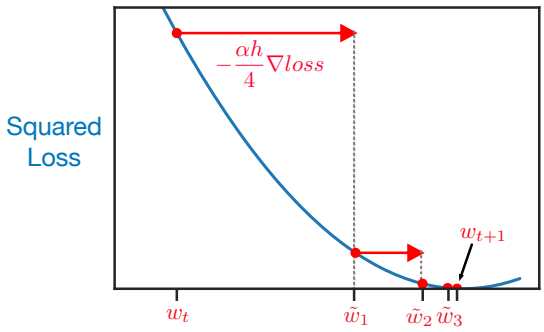

Figure 3.1: Sub-divide the one update into 4 steps and perform 4 intermediate weight updates. The illustration is in one dimensional, and the blue curve is the squared loss surface.

The k updates can be done sequentially through a loop, but we are only interested in the limit of this gradient procedure. By taking $k \to \infty$ and using the fact that $\lim_{k\to\infty}(1 + z/k)^k = \exp(z)$, we obtain:

$$\lim_{k\to\infty} \frac{1 - \left(1 - \frac{\alpha h}{k}\boldsymbol{x}^\top\boldsymbol{x}\right)^k}{\boldsymbol{x}^\top\boldsymbol{x}}(y - \boldsymbol{x}^\top\boldsymbol{w}_t)\boldsymbol{x}$$
$$= \frac{1 - \exp(-\alpha h\boldsymbol{x}^\top\boldsymbol{x})}{\boldsymbol{x}^\top\boldsymbol{x}}(y - \boldsymbol{x}^\top\boldsymbol{w}_t)\boldsymbol{x}.$$

The limit of this gradient procedure as the step-size parameter becomes infinitesimal allows us to generalize (3.4) to

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1 - \exp(-\alpha h_t\boldsymbol{x}_t^\top\boldsymbol{x}_t))}{\boldsymbol{x}_t^\top\boldsymbol{x}_t}(y_t - \boldsymbol{x}_t^\top\boldsymbol{w}_t)\boldsymbol{x}_t, \tag{3.5}$$

where we can use any non-negative continuous numbers as importance weights. In addition, the limit of this gradient procedure only requires an additional computation of an exponential function, and only one update to the weight vector in each time step t.

So far, we have described a technique that we call sliding-step and is introduced by Karampatziakis and Langford (2011) in the supervised learning setting. The picture figure 1.1(b) captures the idea: sub-dividing the one update $\boldsymbol{w}_t + \alpha h_t(y_t - \boldsymbol{x}_t^\top\boldsymbol{w}_t)\boldsymbol{x}_t$ into an infinite number of steps and doing them in sequence, recomputing the gradient at each step. Because the gradient is recomputed in every intermediate steps and the gradients will tend to 0 as the predictions of every intermediate steps tend to y, the estimate after an update $\boldsymbol{x}_t^\top\boldsymbol{w}_{t+1}$ will never overshoot y_t. This has been shown in Karampatziakis and Langford (2011).

The sliding-step algorithm (3.5) empirically outperforms the standard algorithm (3.1) in terms of the quality after a parameter search and in terms of the robustness of the parameter search (Karampatziakis and Langford, 2011). The improvement makes some intuitive sense. The outcome of the sliding-step technique is this expression:

$$\frac{1 - \exp(-\alpha h\boldsymbol{x}^\top\boldsymbol{x})}{\boldsymbol{x}^\top\boldsymbol{x}} \tag{3.6}$$

replacing αh. Here are two ideas that merged. For the ease of writing, let $\delta = y - \boldsymbol{x}^\top\boldsymbol{w}_t$ and $\|\boldsymbol{x}\|^2 = \boldsymbol{x}^\top\boldsymbol{x}$. First of all, the expression (3.6) contains the $\frac{1}{\|\boldsymbol{x}\|^2}$,

which normalizes the gradient since $\big(1-\exp(\cdot)\big)\delta\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|^2}$ of (3.5) replaces $(\cdot)\delta\boldsymbol{x}$ of (3.1). Secondly, the expression (3.6) truncates large αh. This truncation effect is best illustrated by considering

$$\frac{\min(\alpha h\,\|\boldsymbol{x}\|^2, 1)}{\|\boldsymbol{x}\|^2}, \tag{3.7}$$

which upper bounds (3.6) (see figure 1.2). For $\alpha h \geq \frac{1}{\|\boldsymbol{x}\|^2}$, (3.7) truncates αh to $\frac{1}{\|\boldsymbol{x}\|^2}$. For $\alpha h < \frac{1}{\|\boldsymbol{x}\|^2}$, (3.7) is approximately αh. Altogether, (3.6) bounds the size of the update to the weight vector.

3.2 Extending the Sliding-step Technique to TD Learning

We now transition from the realm of supervised learning to TD learning. Let $v(S_t, \boldsymbol{w}), \boldsymbol{w} \in \mathbb{R}^n$ be a function that approximates $v_\pi(S_t)$, the true value of the state $S_t \in \mathcal{S}$ under the policy π. If we have the values of v_π, then we can employ stochastic gradient descent in the supervised learning setting to update the weight vector:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha(v_\pi(S_t) - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t,$$

where $\boldsymbol{x}_t^\top \boldsymbol{w}_t$ is an estimate of $v_\pi(S_t)$ at time step t.

In TD learning, we do not assume to have access to the values of v_π. However, we have access to the one-step bootstrapping target $R_{t+1} + \boldsymbol{x}(S_{t+1})^\top \boldsymbol{w}_t$, and compute this quite easily since we have the reward, the feature vector of the next state, and the current weight vector. If we replace $v_\pi(S_t)$ by the one-step bootstrapping target, then we end up with TD(0) [1]:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \underbrace{\left(R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t\right)}_{\delta_t} \boldsymbol{x}_t, \tag{3.8}$$

where δ_t is the TD error. The update rule (3.8) is similar to the linear rule seen in the supervised learning setting, where an error multiples the $\boldsymbol{x}$.

[1] We omit (0) from now on since all TD algorithms are in the one-step case

Upon first glance, the extension seems almost straight forward. Recall the sliding-step idea: subdivide an update into k steps, where each step has step-size parameter of $\frac{\alpha}{k}$. In each step, we recompute the gradient, and then take k to infinity. We could do the same to the TD update. However, just like TD, the bootstrapping target also depends on the current value of the weight vector. Because the weight vector appears in both the target of the update as well as the prediction, we have two choices to explore and each will impact the extension differently. We can compute the intermediate weight via the semi-gradient or the full-gradient way:

1. semi-gradient: consider only part of the gradient ignoring the effect of changing the weight in the target.

2. full-gradient: consider the effect of changing the weight in both the target and prediction.

The third choice, which we will not explore in this thesis is to change only the weight vector of the target.

3.3 The Sliding-step TD Algorithms

In this section, we apply both the semi-gradient and full-gradient approach to both TD (Sutton and Barto, 1998, 2018) and residual-gradient TD (Baird, 1995).

Since TD (3.8) is a semi-gradient algorithm (Sutton and Barto, 2018), it seems natural to apply the semi-gradient approach to TD. This entails swapping in the intermediate weight vector only in the prediction and holding the weight vector in the target $R_{t+1} + \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t$ fixed throughout all k steps:

$$\tilde{\boldsymbol{w}}_1 = \boldsymbol{w}_t + \frac{\alpha}{k}(R_{t+1} + \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t$$
$$\tilde{\boldsymbol{w}}_2 = \tilde{\boldsymbol{w}}_1 + \frac{\alpha}{k}(R_{t+1} + \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boxed{\boldsymbol{x}_t^\top \tilde{\boldsymbol{w}}_1})\boldsymbol{x}_t$$

After k steps, we obtain:

$$\tilde{\boldsymbol{w}}_k = \boldsymbol{w}_t + \frac{1 - \left(1 - \frac{\alpha}{k}\boldsymbol{x}_t^\top \boldsymbol{x}_t\right)^k}{\boldsymbol{x}_t^\top \boldsymbol{x}_t}\delta_t \boldsymbol{x}_t.$$

By taking $k \to \infty$, we obtain sliding-step TD:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1 - \exp(-\alpha \boldsymbol{x}_t^\top \boldsymbol{x}_t)}{\boldsymbol{x}_t^\top \boldsymbol{x}_t} \delta_t \boldsymbol{x}_t. \tag{3.9}$$

The expression, $\beta(\alpha; S_t) \doteq (1 - \exp(-\alpha \boldsymbol{x}_t^\top \boldsymbol{x}_t))$ is a function of α given S_t, and it replaces the usual step-size parameter α in TD (3.8). If either $\alpha = 0$ or $\boldsymbol{x}^\top \boldsymbol{x} = 0$, $\beta = 0$ and no update is made in that step. If $\alpha \boldsymbol{x}^\top \boldsymbol{x} \to \infty$, $\beta = 1$. Therefore, β is bounded between 0 and 1. If the parameter α is a constant, then $\beta(\alpha; s)$ is a constant with respect to state s. However, $\beta(\alpha; s)$ is different for every $s \in \mathcal{S}$.

3.4 The Sliding-step Residual-gradient TD Algorithm

Residual-gradient TD is a gradient algorithm, which performs gradient descent on the Mean Squared Bellman Error (2.12), and it is defined by the following stochastic update,

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha(R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\Delta_t, \tag{3.10}$$

where $\Delta_t = \boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1}$. It seems natural to apply the full-gradient approach to residual-gradient TD, which entails swapping in the intermediate weight vector in both the target and the prediction throughout all k steps:

$$\begin{aligned}
\tilde{\boldsymbol{w}}_1 &= \boldsymbol{w}_t + \frac{\alpha}{k}(R_{t+1} + \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\Delta_t \\
\tilde{\boldsymbol{w}}_2 &= \tilde{\boldsymbol{w}}_1 + \frac{\alpha}{k}(R_{t+1} + \boxed{\boldsymbol{x}_{t+1}^\top \tilde{\boldsymbol{w}}_1} - \boxed{\boldsymbol{x}_t^\top \tilde{\boldsymbol{w}}_1})\boldsymbol{x}_t.
\end{aligned}$$

After k steps, we obtain:

$$\tilde{\boldsymbol{w}}_k = \boldsymbol{w}_t + \frac{1 - (1 - \frac{\alpha h}{k} \Delta_t^\top \Delta_t)^k}{\Delta_t^\top \Delta_t} \delta_t \Delta_t,$$

By taking $k \to \infty$, we obtain sliding-step residual-gradient TD:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1 - \exp(-\alpha \Delta_t^\top \Delta_t)}{\Delta_t^\top \Delta_t} \delta_t \Delta_t.$$

3.5 A Discussion on the Extension

As it turns out, it is not a good idea to apply the full-gradient approach to a semi-gradient algorithm like TD and semi-gradient approach to a gradient algorithm like residual-gradient TD. In order to discuss the problem, we first derive the following algorithms following the procedures in section 3.3 and section 3.4.

First of all, if we apply the full-gradient approach to TD (3.8), where the intermediate weight vector swaps into both the target and prediction:

$$\tilde{\boldsymbol{w}}_1 = \boldsymbol{w}_t + \frac{\alpha}{k}(R_{t+1} + \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t$$
$$\tilde{\boldsymbol{w}}_2 = \tilde{\boldsymbol{w}}_1 + \frac{\alpha}{k}(R_{t+1} + \boxed{\boldsymbol{x}_{t+1}^\top \tilde{\boldsymbol{w}}_1} - \boxed{\boldsymbol{x}_t^\top \tilde{\boldsymbol{w}}_1})\boldsymbol{x}_t,$$

then after k steps, and by taking $k \to \infty$, we obtain:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1 - \exp(-\alpha \Delta_t^\top \boldsymbol{x}_t)}{\Delta_t^\top \boldsymbol{x}_t}\delta_t \boldsymbol{x}_t. \tag{3.11}$$

Secondly, if we apply the semi-gradient approach to residual-gradient TD (3.10), where the intermediate weight vector swaps into only the prediction:

$$\tilde{\boldsymbol{w}}_1 = \boldsymbol{w}_t + \frac{\alpha}{k}(R_{t+1} + \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\Delta_t$$
$$\tilde{\boldsymbol{w}}_2 = \tilde{\boldsymbol{w}}_1 + \frac{\alpha}{k}(R_{t+1} + \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boxed{\boldsymbol{x}_t^\top \tilde{\boldsymbol{w}}_1})\Delta_t,$$

then after k steps, and by taking $k \to \infty$, we obtain:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{1 - \exp(-\alpha \Delta^\top \boldsymbol{x}_t)}{\Delta^\top \boldsymbol{x}_t}\delta_t \Delta_t. \tag{3.12}$$

Both (3.11) and (3.12) have the following expression,

$$\frac{1 - \exp(-\alpha c)}{c}, \quad c = (\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1})^\top \boldsymbol{x}_t.$$

Given S_t, S_{t+1}, $\frac{1-\exp(-\alpha c)}{c}$ is a function of $\alpha \in [0, \infty)$:

$$\text{case 1: } c = 0, \qquad \frac{1 - \exp(-\alpha c)}{c} \text{ is undefined}$$
$$\text{case 2: } c > 0, \qquad 0 < \frac{1 - \exp(-\alpha c)}{c} \leq \frac{1}{c}$$
$$\text{case 3: } c < 0, \qquad \frac{1 - \exp(-\alpha c)}{c} \geq \alpha,$$

where $|\cdot|$ denote the absolute value. For case 3, $\alpha \in [0, \infty)$ and $c < 0$, therefore, $-\alpha c > 0$. The exponential function, $\exp(z)$ for $z > 0$ is bounded below by its first order Taylor series expansion,

$$\exp(z) \geq 1 + z$$
$$1 - \exp(z) \leq -z.$$

Combining everything together,

$$\frac{1}{-|c|}\Big(1 - \exp(-\alpha|c|)\Big) \geq -\alpha|c|\frac{1}{-|c|}.$$

The expression $\frac{1-\exp(-\alpha c)}{c}$ of case 3 grows exponentially large, which circles back to the same problem of making large updates. Because of case 1 and 3, we eliminate algorithms (3.11) and (3.12) from further experiments.

As demonstrated thus far, extending the sliding-step technique from supervised learning to TD learning is not trivial. In supervised learning, the data are i.i.d where each example can be processed in any order while in temporal difference learning, the data are sequential. By exploiting the sequential nature of the data through bootstrapping, the target of TD learning algorithms also depend on the weight vector. This has led to many choices in terms of extension.

Chapter 4

Analysis of the Sliding-step TD Algorithm

This chapter concerns the second contribution of this thesis. We show that the sliding-step TD in the tabular setting with on-policy training converges with probability 1. In addition, we show how (3.9) compares with the tabular setting. We leave the formal analysis of (3.9) for future research, but we show how it performs empirically in the next chapter.

4.1 The Tabular Setting

For the rest of this chapter, we consider a data stream of the form: $S_t, R_{t+1}, S_{t+1}, \ldots$, where $t = 0, 1, \ldots$. We use $\alpha_s(t) = \alpha(t; s)$ interchangeably throughout this section, and define $\kappa_s : \mathcal{S} \to [0, \infty)$. Let

$$\beta(\alpha_s(t); s) = 1 - \exp(-\alpha_s(t)\kappa_s) \in [0, 1]$$

is a separate sequence for each state $s \in \mathcal{S}$, and they are treated like step-size parameters in the following analysis.

We begin the analysis by defining a table of state values written in vector form, whose component is updated independently and asynchronously. After t time steps, we have a vector of state values $V(t) \in \mathbb{R}^{|\mathcal{S}|}$, with component $V_s(t), s \in \mathcal{S}$, which is updated according to:

$$V_s(t+1) = V_s(t) + \frac{\beta(\alpha_s(t); s)}{\kappa_s}[R_{t+1} + \gamma V_{s'}(t) - V_s(t)], \tag{4.1}$$

where the next state s' transitions from the current state $s \in \mathcal{S}$ according to the transitional probability $p(s'|s)$. The immediate reward R_{t+1} is obtained after transitioning from state s to s'. We call (4.1) the tabular sliding-step TD, which is analogous to tabular TD. Like tabular TD, only the current state's value is updated while all other state values remain unchanged. Tabular sliding-step TD fits the asynchronous model of Bertsekas (1982) and Bertsekas and Tsitsiklis (1989). Together with the asynchronous model and stochastic approximation, Tsitsiklis (1994) prove the convergence of Q-learning with probability 1, where the same proof can be used to prove the convergence of on-line tabular TD (Tsitsiklis, 1994). For the rest of this section, we show how tabular sliding-step TD (4.1) fits into the proof of Tsitsiklis (1994).

Theorem 4.1.1. *$V_s(t)$ defined by the update rule (4.1) converges to $v_\pi(s)$ with probability 1 for all $s \in \mathcal{S}$ in the case of*

(1) $\gamma < 1$ or

(2) $\gamma = 1$, and the policy π is a proper stationary policy

under the condition that $\alpha_s(t)$ is $\mathcal{F}(t)-$measurable [1] for every $s \in \mathcal{S}$ and t, and that the sequence $(\alpha_s(t))$, one for each state s satisfies:

$$\sum_{t=0}^{\infty} \alpha_s(t) = \infty \quad w.p.\ 1$$
$$\sum_{t=0}^{\infty} \alpha_s^2(t) < \infty \quad w.p.\ 1,$$

Proof. We show that the random sequence $(\beta(\alpha_s(t); s))$, one for every state $s \in \mathcal{S}$ also satisfies the following conditions,

$$\sum_{t=0}^{\infty} \beta(\alpha_s(t); s) = \infty \quad \text{w.p. } 1 \tag{4.2}$$
$$\sum_{t=0}^{\infty} \beta^2(\alpha_s(t); s) < \infty \quad \text{w.p. } 1. \tag{4.3}$$

[1] $\mathcal{F}(t)$ represent the history of the algorithm up to the t-th iteration: $S_0, R_1, \alpha_{S_0}(0), V(0), S_1, R_1, \ldots, S_t$. A random variable that is said to be $\mathcal{F}(t)-$measurable means that the random variable is completely determined by the history represented by $\mathcal{F}(t)$

Then, the rest follows from Tsitsiklis (1994) and the iterate $V(t)$ converges to $\boldsymbol{v}_\pi$.

It is reasonable to set $\alpha(\cdot; s) > 0$ when we update V_s and $\alpha(\cdot; s) = 0$ when V_s remains unchanged because $\beta(\alpha_s(t); s) = 0$ whenever $\alpha_s(t) = 0$. Let $X_n = \sum_{t=0}^{n} 1 - \exp(-\alpha_s(t)\kappa_s)$ and the random sequence $X_n, X_{n+1}, \cdots$ is monotonically increasing (i.e., $X_n \leq X_{n+1}$) because $1-\exp(-x)$ is a monotonically increasing function and to get to X_{n+1}, we add something nonnegative to X_n. We show that $\lim_{n\to\infty} X_n = \sum_{t=0}^{\infty} 1 - \exp(-\alpha_s(t)\kappa_s) = \infty$ by using $\exp(-x) \leq 1 + x^2 - x$ for $x \in [0, \infty)$,

$$
\begin{aligned}
1 - \exp(-\kappa_s\alpha_s(t)) &\geq \kappa_s\alpha_s(t) - \kappa_s^2\alpha_s^2(t) \\
\sum_{t=0}^{n} 1 - \exp(-\alpha_s(t)\kappa_s) &\geq \kappa_s \sum_{t=0}^{n} \alpha_s(t) - \kappa_s^2 \sum_{t=0}^{n} \alpha_s^2(t) \\
\lim_{n\to\infty} X_n &\geq \kappa_s \lim_{n\to\infty} \sum_{t=0}^{n} \alpha_s(t) - \kappa_s^2 \lim_{n\to\infty} \sum_{t=0}^{n} \alpha_s^2(t) \\
&\geq \kappa_s \lim_{n\to\infty} \sum_{t=0}^{n} \alpha_s(t) - \kappa_s^2 C_1 = \lim_{n\to\infty} \left(\kappa_s \sum_{t=0}^{n} \alpha_s(t) - \kappa_s^2 C_1 \right) = \infty
\end{aligned}
$$

Ergo, $(\beta(\alpha_s(t); s))$ satisfies (4.2).

Let $Y_n = \sum_{t=0}^{n} (1 - \exp(-\kappa_s\alpha_s(t)))^2$ and the random sequence $Y_n, Y_{n+1}, \cdots$ is monotonically increasing. We show that $\lim_{n\to\infty} Y_n = \sum_{t=0}^{\infty} (1-\exp(-\alpha_s(t)\kappa_s))^2 < C$ for some deterministic constant C by using $\exp(-x) \geq 1 - x$ for $x \in [0, \infty)$

$$
\begin{aligned}
(1 - \exp(-\kappa_s\alpha_s(t))) &\leq (\kappa_s\alpha_s(t)) \\
(1 - \exp(-\kappa_s\alpha_s(t)))^2 &\leq (\kappa_s\alpha_s(t))^2 \\
\sum_{t=0}^{n} (1 - \exp(-\kappa_s\alpha_s(t)))^2 &\leq \kappa_s^2 \sum_{t=0}^{n} \alpha_s^2(t) \\
\lim_{n\to\infty} Y_n &\leq \lim_{n\to\infty} \kappa_s^2 \sum_{t=0}^{n} \alpha_s^2(t) < \infty
\end{aligned}
$$

Ergo, $(\beta(\kappa_s\alpha_s(t)))$ satisfies (4.3)

Since κ_s is a constant given state s, and so is $1/\kappa_s$, the scaled sequence $\left(\frac{\beta(\kappa_s\alpha_s(t))}{\kappa_s} \right)$ also satisfies condition (4.2) and (4.3). □

One way to build the sequence, $(\beta(\kappa_s\alpha_s(t)))$ incrementally is to keep a list of the state visitations. For instance, after having observed the state s at time

step t, we increment the state visitation count, $n(t;s)$ by 1 and set $\alpha(t;s)$ to be $\frac{1}{n(t;s)}$. Because $n(t;s)$ grows by 1 each time state s is visited, the sequence, $1, 1/2, 1/3, \cdots$ clearly satisfies the two conditions.

Alternatively, we can set $\alpha(t) = \frac{1}{t}$ for all state $s \in \mathcal{S}$, where $t = 1, 2, \ldots$. For every state s, there will be a sub-sequence of $\frac{1}{t}$, where t corresponds to the time steps when s is visited. However, $\frac{1}{t}$ is not the same as $\frac{1}{n(t;s)}$, and $\alpha(t)$ may not satisfy the two necessary conditions for convergence. For instance, state s occurs at time $t = 10, 100, 1000 \ldots$, and therefore, the geometric series $\sum_{k=0}^{\infty} \frac{1}{10 \times 10^k} = \frac{1}{9} < \infty$, and hence violates condition (4.2).

4.2 The Linear Function Approximation Setting

The tabular sliding-step TD is not a practical algorithm, but it is useful in comparison with the linear version. Without further ado, we focus our attention on a version of sliding-step TD (3.9) stated as follows,

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \left(\frac{1 - \exp(-\alpha(t) \boldsymbol{x}_t^\top \boldsymbol{x}_t)}{\boldsymbol{x}_t^\top \boldsymbol{x}_t} \right) [R_{t+1} + \boldsymbol{x}_t^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t] \boldsymbol{x}_t, \tag{4.4}$$

where the α is replaced with $\alpha(t)$ instead. (4.4) differs from tabular sliding-step TD (4.1) in the sense that the former generalizes while the latter does not. In the linear function approximation setting, the state values are linear combinations of the feature vector $\boldsymbol{x} \in \mathbb{R}^n$ and a weight vector $\boldsymbol{w} \in \mathbb{R}^n$. The dimension n can equal $|\mathcal{S}|$ or less than $|\mathcal{S}|$. Either way, whenever the weight vector changes via (4.4), the change may affect the values of many states. Such generalization of the state values based on one state-to-state transition may speed up learning but also makes the analysis more difficult. We leave the formal analysis of sliding-step TD (4.4) for future research. For the remainder of this chapter, we discuss how the parameter α can influence the behavior of sliding-step TD. In the next chapter, we show empirically how the choice of feature vectors can also influence the behavior of sliding-step TD.

4.2.1 Special Cases

If the feature vectors are all of the same magnitude (i.e., $\boldsymbol{x}(s)^\top \boldsymbol{x}(s) = C$ for all $s \in \mathcal{S}$) and sequence $(\alpha(t))$ satisfy the step-size conditions (Robbins and Monro, 1951), then the expression,

$$\frac{1 - \exp(-\alpha(t)\boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)} = \frac{1 - \exp(-\alpha(t)C)}{C} \tag{4.5}$$

is a scaled sequence that also satisfy the step-size conditions by similar argument made in the proof of theorem 4.1.1. Then, sliding-step TD deduces to TD, and the rest follows from the results of Tsitsiklis and Van Roy (1997); $\boldsymbol{w}_t$ converges to the fixed point of TD with probability 1. An example of $(\alpha(t))$ that would allow (4.5) to satisfy the step-size conditions is the diminishing sequence $\frac{1}{t}$, $t = 1, 2 \ldots$

Some examples of feature vectors whose magnitude are all the same include the basis unit vectors and normalized feature vectors (i.e., $\frac{\boldsymbol{x}(s)}{\|\boldsymbol{x}(s)\|}$ for all $s \in \mathcal{S}$). In both cases, $C = 1$ for all $s \in \mathcal{S}$. Regardless of the dimension n equals the $|\mathcal{S}|$ or less than $|\mathcal{S}|$, sliding-step TD simplifies to TD:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + (1 - \exp(-\alpha(t)))(R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t$$

with a particular sequence of step-size parameters $1 - \exp(-\alpha(t))$.

4.2.2 General Case

The general case involves feature representation $\boldsymbol{x}(s) \in \mathbb{R}^n$, and n can equal $|\mathcal{S}|$ or less than $|\mathcal{S}|$. We consider a constant $\alpha \in [0, \infty)$, and the expression $\frac{1-\exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)}$ depends on the choice of α.

When α is small, we expect sliding-step TD to behave similarly to TD. This is because

$$\frac{1 - \exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)} \approx \frac{\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s)}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)} = \alpha$$

for all state $s \in \mathcal{S}$. We see this by first fixing s and examine the first order Taylor series expansion of $\beta(\alpha; s) = 1 - \exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))$ around $\alpha = 0$:

$$\beta(\alpha; s) = \beta(0; s) + \beta^{(1)}(0; s)(\alpha - 0) + R_1 = \alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s) + R_1,$$

where $\beta^{(i)}$ denote the ith order derivative of function β with respect to α. For $0 < \xi < \alpha$,

$$R_1 = \frac{\beta^{(2)}(\xi)\alpha^2}{2!} = \frac{-(\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))^2 \exp(-\xi \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{2}$$
$$|R_1| = \frac{(\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))^2 \exp(-\xi \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{2} \leq \frac{(\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))^2}{2}.$$

Thus there exist a $\delta > 0$ such that

$$|R_1| \leq C|\alpha^2| \text{ for all } |\alpha - 0| < \delta,$$

where $C = \frac{(\boldsymbol{x}(s)^\top \boldsymbol{x}(s))^2}{2}$. Therefore, the reminder,

$$R_1 = O(\alpha^2), \ \ \alpha \to 0. \tag{4.6}$$

On the other hand, $\alpha \to \infty$, $\frac{1-\exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)} \to \frac{1}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)}$.

If we consider (4.4) with a sequence of diminishing $\alpha(t)$ (i.e., $\alpha(t) \to 0$), then at some point, all the expression $\frac{1-\exp(-\alpha(t) \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)} \approx \alpha$ due to (4.6). We would expect sliding-step TD to behave similarly to TD.

For a constant α that is not sufficiently small [2], the expression $\frac{1-\exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)}$ is a different constant in $[0, \infty)$ for different feature vectors. If two states are represented by the same feature vector, then the expression would be the same. Although the expression only scales the steps, and the direction of the update determined by $\delta_t \boldsymbol{x}_t$ is the same as TD, the iterates $\boldsymbol{w}_t$ are not averaged the same way as TD. The stochastic update of the sliding-step TD does not average to the following expected update,

$$\mathbb{E}_{\boldsymbol{d}_\pi}\left[\underbrace{\frac{1-\exp(-\alpha \boldsymbol{x}_t^\top \boldsymbol{x}_t)}{\boldsymbol{x}_t^\top \boldsymbol{x}_t}}_{g(\alpha; S_t)}(R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w} - \boldsymbol{x}_t^\top \boldsymbol{w})\boldsymbol{x}_t\right]$$
$$= \underbrace{\mathbb{E}_{\boldsymbol{d}_\pi}\left[g(\alpha; S_t) R_{t+1} \boldsymbol{x}_t\right]}_{\mathbf{b}} - \underbrace{\mathbb{E}_{\boldsymbol{d}_\pi}\left[g(\alpha; S_t)\boldsymbol{x}_t(\boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1})^\top\right]}_{\mathbf{A}} \boldsymbol{w}$$

for some fixed $\boldsymbol{w}$. The expectation is evaluated w.r.t. the steady-state distribution $\boldsymbol{d}_\pi$, which we assume exists and is positive for all states. Let $\mathbf{G} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$

[2] By sufficiently small, we mean (4.6)

be a diagonal matrix whose diagonal entries are $g(\alpha; s)$ and $\mathbf{D}_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ be another diagonal matrix whose diagonal entries are $\boldsymbol{d}_\pi(s)$. Writing out the $\mathbf{A}$ and $\mathbf{b}$ in matrix and vector forms as follows,

$$\begin{aligned}
\mathbf{A} &= \mathbb{E}_{\boldsymbol{d}_\pi}\left[g(\alpha; S_t)\boldsymbol{x}_t(\boldsymbol{x}_t - \gamma\boldsymbol{x}_{t+1})^\top\right] \\
&= \sum_s \boldsymbol{d}_\pi(s)\mathbb{E}_{\boldsymbol{d}_\pi}\left[g(\alpha; s)|S_t = s\right]\mathbb{E}_{\boldsymbol{d}_\pi}\left[\boldsymbol{x}_t(\boldsymbol{x}_t - \gamma\boldsymbol{x}_{t+1})^\top|S_t = s\right] \\
&= \sum_s \boldsymbol{d}_\pi(s)g(\alpha; s)\boldsymbol{x}(s)\Big(\boldsymbol{x}(s) - \sum_{s'}\gamma[\mathbf{P}_\pi]_{ss'}\boldsymbol{x}(s')\Big) \\
&= \boldsymbol{X}^\top\mathbf{G}\mathbf{D}_\pi(\mathbf{I} - \gamma\mathbf{P}_\pi)\boldsymbol{X},
\end{aligned}$$

$$\begin{aligned}
\mathbf{b} &= \mathbb{E}_{\boldsymbol{d}_\pi}\left[g(\alpha; S_t)R_{t+1}\boldsymbol{x}_t\right] \\
&= \boldsymbol{X}^\top\mathbf{G}\mathbf{D}_\pi\boldsymbol{r}_\pi.
\end{aligned}$$

If $\mathbf{A}$ has a full set of eigenvalues all of whose real parts are positive, then we can solve $\mathbf{A}\boldsymbol{w} = \mathbf{b}$ (Sutton, 1988; Tsitsiklis and Van Roy, 1997; Sutton et al., 2016) and obtain,

$$\begin{aligned}
\boldsymbol{w}' &= \mathbf{A}^{-1}\mathbf{b} \\
&= (\boldsymbol{X}^\top\mathbf{G}\mathbf{D}_\pi\boldsymbol{X} - \gamma\boldsymbol{X}^\top\mathbf{G}\mathbf{D}_\pi\mathbf{P}_\pi\boldsymbol{X})^{-1}\boldsymbol{X}^\top\mathbf{G}\mathbf{D}_\pi\boldsymbol{r}_\pi. \qquad (4.7)
\end{aligned}$$

In general, $\mathbf{A}$ may not have a full set of eigenvalues all of whose real parts are positive (e.g., set $\alpha = 4$ in the following two-state Markov chain experiment). However, the iterates of sliding-step TD also does not oscillate around $\boldsymbol{w}'$. This is best illustrated via the following experiment involving a two-state Markov chain,

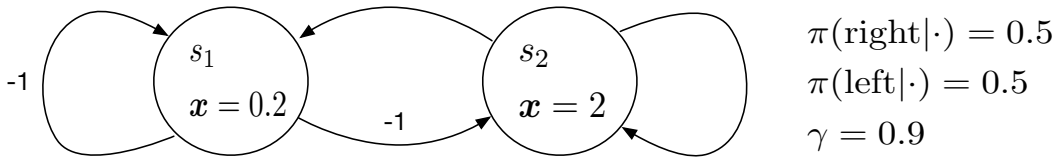

Figure 4.1: Two states random walk without a terminal state

State 1 is represented by a scalar value of 0.2 while state 2 is represented by

a scalar value of 2. In this case, the feature matrix is of rank 1, smaller than the number of states, and the true state values cannot be represented exactly.

The probability of taking the left or right action is 0.5. Taking the left action in any state results in a reward of -1 and taking the right action in any state results in a reward of 0. The policy π induces a Markov chain whose process is determined by $\mathbf{P}_\pi = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$. The steady-state distribution can be computed exactly and is $\boldsymbol{d}_\pi = [0.5, 0.5]^\top$. The true state values $\boldsymbol{v}_\pi$ can also be computed exactly and are [-5, -5]$^\top$, where the first component corresponds to state 1 and second component corresponds to state 2.

For a sufficiently small α, sliding-step TD performs similarly to TD. For an α that is not sufficiently small, the iterates of sliding-step TD oscillate in a neighborhood that is different from the fixed point of TD (2.15) and also different from $\boldsymbol{w}'$.

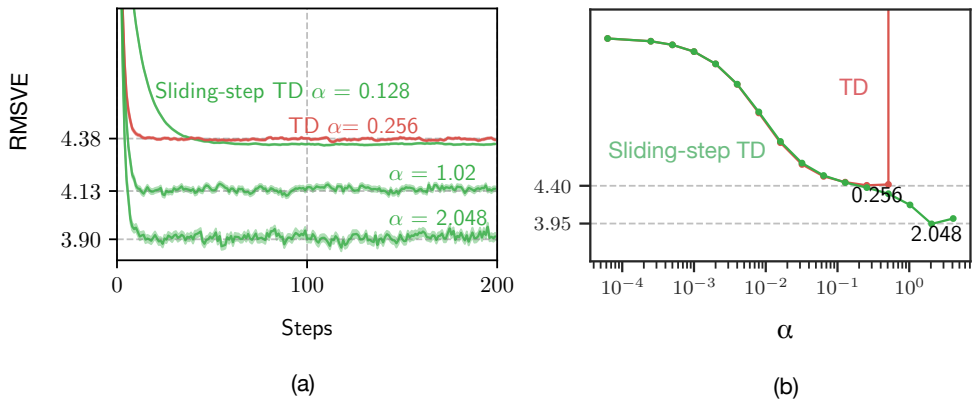

Figure 4.2: Illustration of the fixed point of TD and sliding-step TD. (a) is the learning curves with various α's. (b) is sensitivity graphs where α is searched in powers of 2. The optimal α is found to be 0.256 for TD and 2.048 for sliding-step TD. The RMSVEs for the learning curves are averaged over 500 runs. The RMSVEs for the sensitivity graphs are averages of 200 steps and then averaged over the 500 runs.

Results and Discussion:

In this problem, we are interested in the quality of the solution, by which we mean how closely the state value estimates approximate $\boldsymbol{v}_\pi$ weighted by their respective state visitation. Therefore RMSVE (2.16) is an appropriate performance measure.

As a sanity check, TD's fixed point is -0.59 according to (2.15), which should produce a RMSVE($\bar{\boldsymbol{w}}$) of 4.38. This matched the observations in figure 4.2(a). For $\alpha = 2.048$, the eigenvalue of $\mathbf{A}$ is 0.0977, which is real and positive and $\boldsymbol{w}' = -2.28$ according to (4.7). Thus, the RMSVE($\boldsymbol{w}'$) is 3.22, but we observed the RMSVE($\boldsymbol{w}_t$) of sliding-step TD fluctuating around 3.9 instead, which is statistically different from 3.22. Checking a few more α's, we observed similar mismatch. For $\alpha = 1.02$, RMSVE($\boldsymbol{w}'$) is 3.98, but we observed RMSVE($\boldsymbol{w}_t$) of sliding-step TD fluctuating around 4.13. For $\alpha = 0.128$, RMSVE($\boldsymbol{w}'$) is 4.35, but we observed RMSVE($\boldsymbol{w}_t$) of sliding-step TD fluctuating around 4.38, which was similar to TD. As α decreases, we saw the performance of sliding-step TD matching that of the TD as seen in figure 4.2(b). For a small enough α, we conjecture that sliding-step TD in the long run should behave in an average sense similar to that of (2.14).

4.3 Comparison to Normalized TD Algorithm

Sliding-step TD is similar to another algorithm called the normalized TD (Bradtke, 1994). Both algorithms intend to control the size of the updates. Both algorithms have similar stochastic updates:

$$\text{Normalized TD: } \alpha(R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t) \frac{\boldsymbol{x}_t}{\epsilon + \boldsymbol{x}_t^\top \boldsymbol{x}_t}$$

$$\text{Sliding-step TD: } (1 - \exp(-\alpha \boldsymbol{x}_t^\top \boldsymbol{x}_t))(R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t) \frac{\boldsymbol{x}_t}{\boldsymbol{x}_t^\top \boldsymbol{x}_t},$$

where ϵ is a small, positive number, which can be set to 0 if $\boldsymbol{x}(s)$ are non-zero for all $s \in \mathcal{S}$. We shall compare the two algorithms empirically in the next chapter.

Chapter 5

Experiments with the Sliding-step TD Algorithm

This chapter concerns the third contribution of this thesis. In particular, we present experiments to compare how tabular sliding-step TD differs from tabular TD. Then, we show how feature choices may affect the performance of the linear case: sliding-step TD, TD and normalized TD in the on-policy setting.

5.1 The Goal of the Study

Goal 1: We study how κ_s of the tabular sliding-step TD affect the performance of the algorithm. Tabular TD serves as the baseline for comparison. The simplest way is to set κ_s to be a constant (e.g. $\kappa_s = 1$) for all $s \in \mathcal{S}$, since there is no concept of feature representations for each state. Another choice is set $\kappa_s = \boldsymbol{x}(s)^\top \boldsymbol{x}(s)$ to match the expression $\frac{1-\exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)}$ of sliding-step TD in the linear case.

Goal 2: We study the effect of feature vectors on the performance of sliding-step TD. The evaluation involves two ways of setting the α. The first way is to set α as a constant for all states through out all time steps. The second way is to set $\alpha(t)$ as a diminishing sequence for all states.

5.2 The Domain of the Study

We define the following five-state Markov chain by the following transition and reward matrices,

$$\mathbf{P}_\pi = \begin{bmatrix} 0.42 & 0.13 & 0.14 & 0.03 & 0.28 \\ 0.25 & 0.09 & 0.16 & 0.35 & 0.15 \\ 0.08 & 0.20 & 0.33 & 0.17 & 0.22 \\ 0.36 & 0.05 & 0.00 & 0.52 & 0.07 \\ 0.17 & 0.24 & 0.19 & 0.18 & 0.22 \end{bmatrix}$$

$$R = \begin{bmatrix} -3.62 & 102.1 & 27.92 & 128.44 & 9.7 \\ -56.5 & 31.61 & 104.15 & 143.92 & -4.74 \\ 201.72 & -76.23 & 44.58 & 101.1 & 33.2 \\ 84.7 & 84.61 & 25.01 & 3.25 & 68.32 \\ 92.36 & -53.94 & 0.49 & 24.03 & 204.8 \end{bmatrix}.$$

The transition matrix is exactly like the one in Bradtke and Barto (1996). The reward matrix is arbitrarily generated to include both positive and negative values that cover a large range of numbers. The steady-state distribution of this Markov chain can be computed exactly, and it is $[0.28, 0.14, 0.15, 0.24, 0.19]^\top$. The true state values $\boldsymbol{v}_\pi$ are [378.91, 410.74, 402.62, 391.85, 413.03]$^\top$, where the ith component corresponds to state i and $i = 1, \cdots, 5$.

5.3 The Representations

We represent each state by a vector of real values, transpose each vector, and stack them into a feature matrix. This study involves two sets of feature matrices with linearly independent columns. The first set of matrices $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ have ranks equal the number of the states while the second set of feature matrices $\boldsymbol{X}'_1$ and $\boldsymbol{X}'_2$ have ranks less than the number of the states. Function approximation with the first set of feature representations can represent all the states values $\boldsymbol{v}_\pi$ exactly while the second set of feature representations cannot.

$$\boldsymbol{X}_1 = \begin{bmatrix} -160.59 & -117.66 & 80.84 & -158.6 & 61.56 \\ 77.58 & 210.62 & -128.85 & -29.88 & 54.83 \\ -55.58 & -86.12 & -24.54 & 149.71 & -75.27 \\ 46.78 & -96.32 & 78.13 & -8.75 & -122.62 \\ -92.71 & 33.7 & -31.62 & 1.8 & -115.98 \end{bmatrix}$$

The magnitude of the feature vector $\boldsymbol{x}(s)^\top \boldsymbol{x}(s)$ for states 1 to 5 are 75111.72, 96396.90, 39186.65, 32682.43, 24185.25.

$$\boldsymbol{X}_2 = \begin{bmatrix} 0.31 & 1.55 & 18.51 & -0.6 & 1.22 \\ -1.91 & 1.77 & -2.51 & 0.71 & 2.21 \\ -0.16 & 0.23 & 7.3 & -1.04 & 3.07 \\ 0.04 & 1.95 & 8.6 & 0.23 & 0.04 \\ 0.55 & 0.46 & 33.84 & -1.18 & 1.54 \end{bmatrix}$$

The magnitude of each feature vector for states 1 to 5 are 346.97, 18.47, 63.88, 77.82, 1149.42. The magnitude of feature vectors of $\boldsymbol{X}_2$ are more varying than $\boldsymbol{X}_1$.

$$\boldsymbol{X}'_1 = \begin{bmatrix} -160.59 & 80.84 & 61.56 \\ 177.58 & -128.85 & 54.83 \\ -55.58 & -24.54 & -75.27 \\ 46.78 & 78.13 & -122.62 \\ -92.71 & -31.62 & -115.98 \end{bmatrix}$$

The magnitude of each feature vector for states 1 to 5 are 36113.88, 51143.30, 9356.92, 23328.32, 23046.32.

$$\boldsymbol{X}'_2 = \begin{bmatrix} 0.31 & 18.51 & 1.22 \\ -1.91 & -2.51 & 2.21 \\ -0.16 & 7.3 & 3.07 \\ 0.04 & 8.6 & 0.04 \\ 0.55 & 33.84 & 1.54 \end{bmatrix}$$

The magnitude of each feature vector for states 1 to 5 are 344.2 , 14.83, 62.74, 73.96, 1147.82. Again, the magnitude of feature vectors of $\boldsymbol{X}'_2$ are more varying than $\boldsymbol{X}'_1$.

5.4 The Setup

For a decaying sequence $\alpha(t)$, we chose the equation $\frac{\eta_0}{(1+(t/\tau))}$ from Darken and Moody (1990). The parameters, η_0, τ are constants; η_0 controls the initial $\alpha(t)$, $t < \tau$ indicates the search phase while $t > \tau$ indicates the converge phase. Although $\frac{\eta_0}{1+(t/\tau)}$ offers a wide range of values for $\alpha(t)$, but it also has more parameters to tune. The α and the η_0 are searched in powers of 2 while the τ is searched in increment of 100. To obtain $\alpha(t; s)$ for the tabular setting, we keep track of the state visitation counts $n(t; s)$ up till time step t, and set $\alpha(t; s) = \frac{\eta_0}{1+n(t;s)/\tau}$.

We are interested in the quality of the solution and the speed of the learning in each evaluation. With regard to the quality of the solution, we mean how closely the state value estimates approximate v_π weighted by their respective state visitation. Therefore RMSVE is an appropriate performance measure.

We generate two types of graphs to visually compare the performance of the algorithms. First, we average each RMSVE($\boldsymbol{w}_t$) over the runs, and plot the mean against the steps to generate the learning curve. Second, we sum up all the RMSVE($\boldsymbol{w}_t$) computed in each step of a run, and plot the mean of the runs against each of the α to generate the sensitivity graph. The learning curves help us compare how fast each algorithm learns as well as the quality of the solutions in the long run. The sensitivity graphs help us compare how sensitive is each algorithm towards the changes in the parameter. With regard to the speed of learning, we mean how quickly the RMSVE decays with respect to time steps.

5.5 The Tabular Setting

We show results for Goal 1: the choice of $\kappa_s \in [0, \infty)$ can impact the performance of the tabular sliding-step TD. We can set $\kappa_s = \boldsymbol{x}(s)^\top \boldsymbol{x}(s)$ to match the expression $\frac{1-\exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)}$ of the sliding-step TD. For feature representation $\boldsymbol{X}_1$, tabular sliding-step TD learned slowly. This is because the magnitude of the feature vectors are all very large, and thus each scaling factor $\frac{1}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)}$

is very small for all $s \in \mathcal{S}$. The small scaling factors scale down the size of each adjustment to the state values making learning slower. On the other hand, with a different feature representation $\boldsymbol{X}_2$, tabular sliding-step TD with $\kappa_s = \boldsymbol{x}(s)^\top \boldsymbol{x}(s)$ learned faster. Finally, by setting κ_s to a constant of 1, 2, or 4 for all $s \in \mathcal{S}$, we saw no statistically significant performance difference between tabular sliding-step TD and tabular TD.

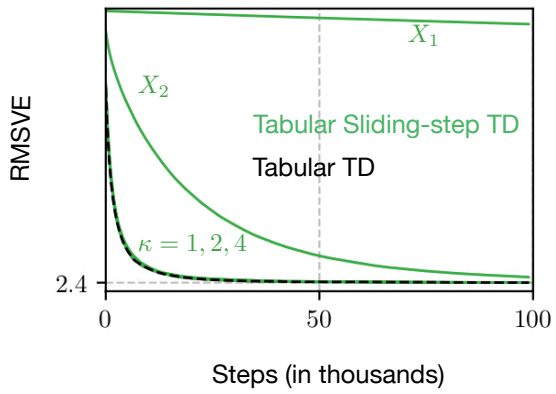

Figure 5.1: Five-state Markov Chain: performance comparison of tabular sliding-step TD with tabular TD. The black dotted line is the learning curve for tabular TD. For label X_1 and X_2, we choose $\kappa_s = \boldsymbol{x}(s)^\top \boldsymbol{x}(s)$. For $\kappa = 1, 2, 4$, κ_s is a constant of value $1, 2$, or 4. Each learning curve has been optimized for η_0, τ where $\alpha(t; s) = \eta_0/(1 + n(t; s)/\tau)$. The RMSVEs are averaged over 50 runs.

5.6 The Linear Function Approximation Setting

We present the results for Goal 2: comparing the performance of sliding-step TD to that of TD and normalized TD. We averaged each RMSVE($\boldsymbol{w}_t$) over the 50 runs and plotted the mean against the steps to generate the learning curve. We applied a moving window of size 1000 along the steps to further smooth the curve and contrast the differences. Second, we summed the 100,000 RMSVE($\boldsymbol{w}_t$) for each run and acquired 50 sums for each α. We plotted the mean of the 50 sums against each of the α to generate the sensitivity graph with respect to α.

Results and Discussions:

In all experiments, sliding-step TD performed in a wider range of α values than TD, and hence providing substantial evidence of robustness in sliding-step TD. Sliding-step TD behaved more like TD than normalized TD. This was expected as for small α, the expression $\frac{1-\exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)} \approx \alpha$.

Sliding-step TD learned faster than TD when the magnitude of the feature vectors $\boldsymbol{x}(s)^\top \boldsymbol{x}(s)$ varied significantly as observed in figure 5.4 and figure 5.5. One possible explanation for this result is due to the expression $\frac{1-\exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)}$, which normalizes the magnitude of feature vectors, and hence bounding the size of the update.

In the case when the magnitude of feature vectors are all large, sliding-step TD still needed a small α or η_0 so not to diverge. With small α's, we found no statistically significant speedup in sliding-step TD. However, sliding-step TD was still more robust than TD. Let us examine figure 5.2(a) in detail. A small α such as 3.0 e-5 would have pushed $1 - \exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))$ towards a value of 1 for all 5 states. An value of 1 is too large, because normalized TD with $\alpha > 5.24$ e-1 already diverged, and normalized TD has a similar expression of $\frac{\alpha}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)}$ compared to the sliding-step TD's expression of $\frac{1-\exp(-\alpha \boldsymbol{x}(s)^\top \boldsymbol{x}(s))}{\boldsymbol{x}(s)^\top \boldsymbol{x}(s)}$. The optimal $\alpha = 5$ e-6 made the expression all approximately 4 e-6 for all 5 states. Sliding-step TD performed similarly to TD with the optimized step-size parameter $\alpha = 4$ e-6.

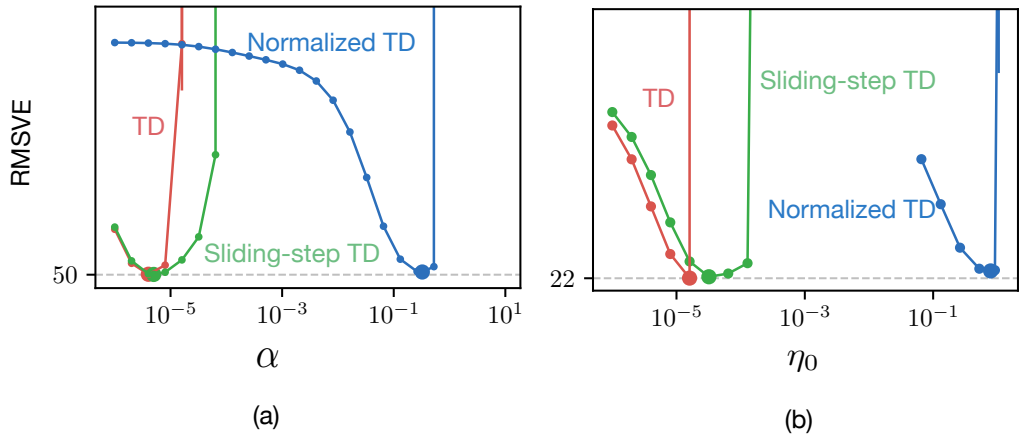

Figure 5.2: Five-state Markov Chain: performance comparison of sliding-step TD, TD and normalized TD with feature matrix $\boldsymbol{X}_1$. (b) is optimized for τ and shown w.r.t. η_0 of $\alpha(t) = \frac{\eta_0}{1+t/\tau}$. The RMSVEs are averages of 100 thousand steps and then averaged over 50 runs. The big dots indicate the optimal parameter values.

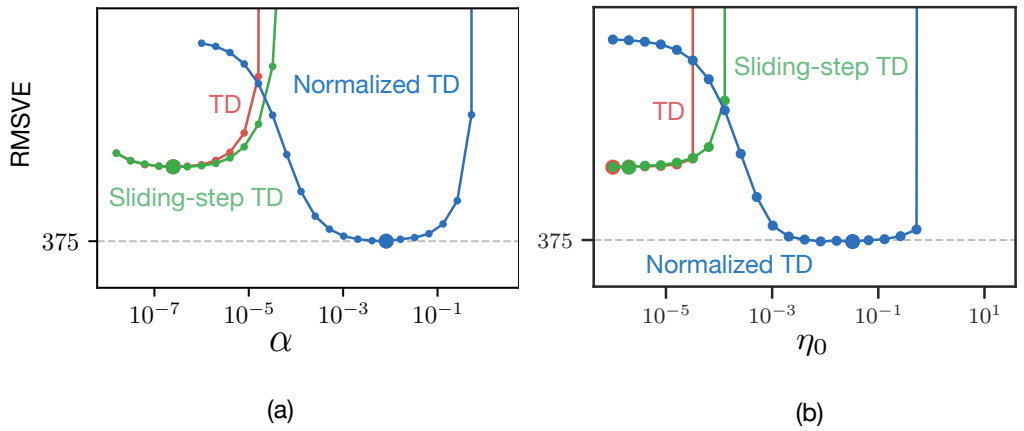

Figure 5.3: Five-state Markov Chain: performance comparison of sliding-step TD, TD and normalized TD with feature matrix $\boldsymbol{X}_1'$. (b) is optimized for τ and shown w.r.t. η_0 of $\alpha(t) = \frac{\eta_0}{1+t/\tau}$. The RMSVEs are averages of 100 thousand steps and then averaged over 50 runs. The big dots indicate the optimal parameter values.

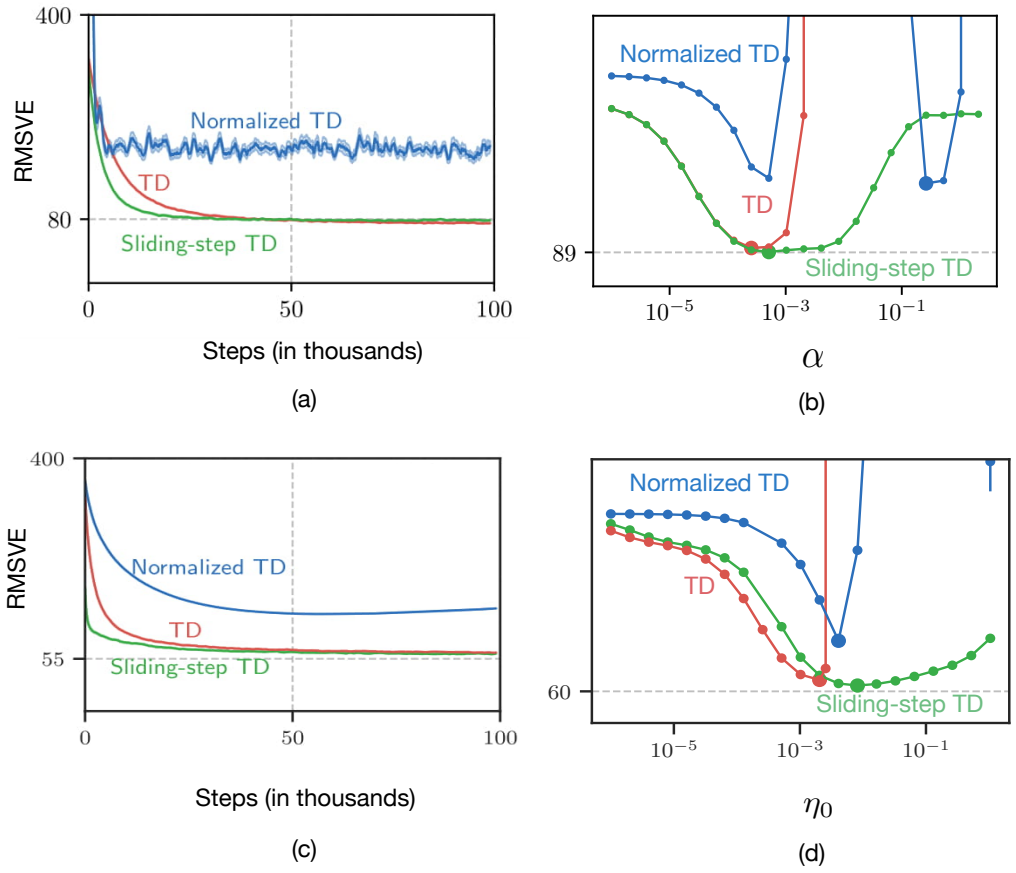

Figure 5.4: Five-state Markov Chain: performance comparison of Sliding-step TD, TD and Normalized TD with feature matrix $\boldsymbol{X}_2$. (a) and (c) are learning curves optimized for α. (b) and (d) are sensitivity graphs optimized for τ and shown w.r.t. η_0 of $\alpha(t) = \frac{\eta_0}{1+t/\tau}$. Row 1 is w.r.t. constant step-size parameter. Row 2 is w.r.t. diminishing step-size parameter. The RMSVEs for the learning curves are averaged over 50 runs. The RMSVEs for the sensitivity graphs are averages of 100 thousand steps and then averaged over 50 runs. The big dots in the sensitivity graphs indicate the optimal parameter values.

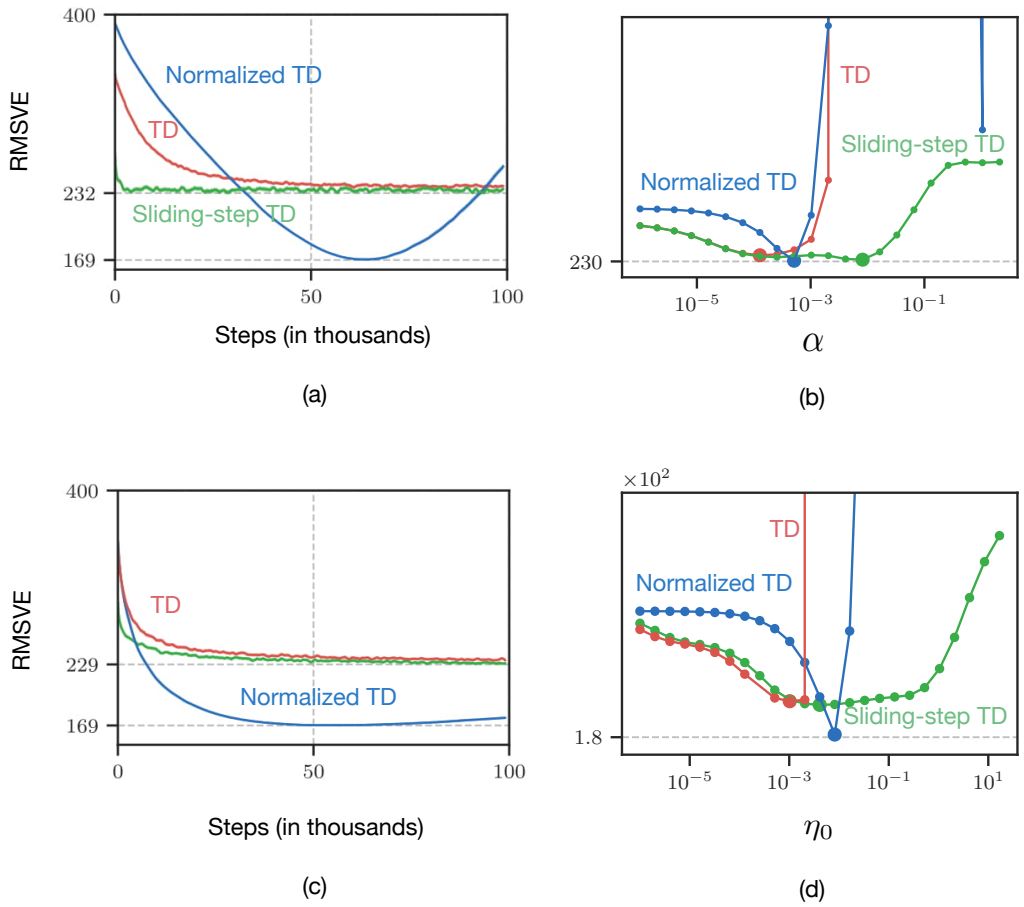

Figure 5.5: Five-state Markov Chain: performance comparison of Sliding-step TD, TD and Normalized TD with feature matrix $\boldsymbol{X}_2'$. (a) and (c) are learning curves optimized for α. (b) and (d) are sensitivity graphs optimized for τ and shown w.r.t. η_0 of $\alpha(t) = \frac{\eta_0}{1+t/\tau}$. Row 1 is w.r.t. constant step-size parameter. Row 2 is w.r.t. diminishing step-size parameter. The RMSVEs for the learning curves are averaged over 50 runs. The RMSVEs for the sensitivity graphs are averages of 100 thousand steps and then averaged over 50 runs. The big dots in the sensitivity graphs indicate the optimal parameter values.

Chapter 6

Emphatic TD and Residual-gradient TD Extensions

This chapter concerns the last contribution of this thesis, where we extend the sliding-step technique to emphatic TD and residual-gradient TD. We compare the performance of sliding-step emphatic TD and sliding-step residual-gradient TD by ways of two experiments.

6.1 The Sliding-step Emphatic TD Algorithm

We consider both the on-policy and off-policy training, and this is because both the emphatic TD and residual-gradient TD are stable and convergent under off-policy training. In addition, the stochastic update of both algorithms have importance sampling ratio and emphasis, which can result in large updates to the weight vector. We present the sliding-step emphatic TD (6.1) and sliding-step residual-gradient TD (6.2) in the off-policy setting. This is because in the on-policy setting, the importance sampling ratios are all ones.

The sliding-step emphatic TD is defined by the following update to its weight vector,

$$\begin{aligned} \boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \frac{1-\exp(-\alpha F_t \rho_t \boldsymbol{x}_t^\top \boldsymbol{x}_t)}{\boldsymbol{x}_t^\top \boldsymbol{x}_t}(R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t)\boldsymbol{x}_t \\ \text{where } \; F_t &= \gamma \rho_{t-1} F_{t-1} + i(S_t) \quad \text{and} \quad F_0 = 1. \end{aligned} \tag{6.1}$$

Some updates of emphatic TD are weighted more than others by a quantity

called the follow-on trace F_t [1]. The follow-on trace is a sum of all the visited state's interest $i(S_t) \in [0, \infty)$ up to step t. F_t can take on large values and as matter of fact, the variance of F_t can be infinite (Sutton et al., 2016). In off-policy training, emphatic TD also has importance sampling ratio. Importance sampling ratio for step t is defined as $\rho_t = \pi(A_t|S_t)/\mu(A_t|S_t)$, and it can be large if $\mu(A_t|S_t)$ is small. Since $\rho_0, \cdots \rho_{t-1}$ are all incorporated inside the follow-on trace, this further exacerbates the problem of having a large F_t. To make matter worse, ρ_t multiplies F_t in the update resulting in an even larger value. If the step-size parameter α is not reduced to small enough value, one update to the weight vector could increase the TD error than reducing it. Sliding-step emphatic TD has F_t and ρ_t incorporated inside the expression:

$$\beta(\alpha; \rho_t, F_t; S_t) = 1 - \exp(-\alpha \rho_t F_t \boldsymbol{x}_t^\top \boldsymbol{x}_t).$$

Hence, bounding the size of the update due to the F_t and ρ_t because $\beta(\alpha; \rho_t, F_t; S_t) \to 1$ as $F_t \rho_t \to \infty$.

6.2 The Sliding-step Residual-gradient TD Algorithm

In the off-policy case, the importance sampling ratio of residual-gradient TD can scale up the size of the update in just one step, increasing the chance for unstable behavior. However, the importance sampling ratio is inside the expression of the sliding-step residual-gradient TD, which is defined by the following stochastic update to its parameter vector,

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \frac{(1 - \exp(-\alpha \rho_t \Delta_t^\top \Delta_t))}{\Delta_t^\top \Delta_t} (R_{t+1} + \gamma \boldsymbol{x}_{t+1}^\top \boldsymbol{w}_t - \boldsymbol{x}_t^\top \boldsymbol{w}_t) \Delta_t$$
$$\text{where } \Delta_t = \boldsymbol{x}_t - \gamma \boldsymbol{x}_{t+1}. \tag{6.2}$$

6.3 1000-state Random Walk

This experiment is in the on-policy setting. We change the number of neighbors in the original problem (Sutton and Barto, 2018). Instead of 100 neighbors, we

[1] For $\lambda = 0$, the follow-on trace is the emphasis. In the more general setting of emphatic TD(λ), these two quantities are not equivalent.

consider 50 and then 20 neighbors. The state space consists of states 1 to 1000 with two terminal states; one positioned left of state 1 and one positioned right of state 1000. The agent always starts in state 500, and it takes a left or right action with equal probability. In the case of 50 neighbors, once committed to an action, agent transitions to one of its 50 neighbors with equal probability. Of course, there is not always 50 neighbors to the left or right of a state, so whatever remains of the 50 neighbors account toward the terminal state. For example, if the current state is 975 and the agent decides to go right, then 25 of the remaining 50 is considered part of the terminal state and the agent transitions to one of the state in the range of 976 to 1000. Thus going right in state 975 will have a probability of 0.25 of terminating on the right. The reward is 0 everywhere except a -1 when transitioning to the left terminal state and a +1 when transitioning to the right terminal state. The discount factor is $\gamma = 1$.

The reason for the modification of this problem from 100 neighbors to 50 and 20 neighbors is for us to observe a larger range of follow-on trace values. Since $\gamma = 1$ and all of the states has interest of 1, then the follow-on trace (i.e., $F_t = F_{t-1} + 1$) grows linearly with the number of time step in an episode. The random walk takes less number of steps to reach the terminal states when it can reach 1 of 100 neighbors versus 1 of 20 neighbors at a time since the former is more likely to reach further to the left or right. However, just how large can the number of steps be before the random walk terminates if we start in state 500? We list out the expected number of steps before termination if the starting state is in state 500. On average, F can take on the following values for a neighbor of 50 and 20,

m	expected number of steps before termination	variance of number of steps before termination
50	309.14	63533.10
20	1785.15	2123450.03

Table 6.1: Expected number of steps before termination in the 1000-states random walk

For each of the above number of neighbors, there is a high probability of

having large F's.

In the following experiments, we will use two feature representations: tile coding and Fourier basis. We run each experiment for 5000 episodes and repeat each independently 30 times. At the end of each episode, an RMSVE($\boldsymbol{w}_\tau$) (τ denote the time when the episode terminates) is computed and plotted in the learning curve with respect to episode. For the sensitivity graph, all 5000 RMSVEs are summed and presented for each α.

With tile coding, we used 50 tilings, and each tiling is offset by four states. We tile 100 states together. The tile coding serves as a basis of our comparison since the feature vectors are a list of 1's and 0's and the number of active tiles will always equal the number of tilings, 50 in this case. Thus, the size of the feature vectors, $\boldsymbol{x}(s)^\top \boldsymbol{x}(s) = 50$, the number of tilings for all states. We will experiment with emphatic TD and sliding-step emphatic TD.

Results and Discussions

Going from 50 neighbors to 20 neighbors, the follow-on trace on average increases. In response to the larger emphasis value, the optimal α shifted from an order of e-5 to e-6. However, sliding-step emphatic TD performed in a wider range of α's than emphatic TD. We saw evidence of robustness again in sliding-step emphatic TD. Similarly, sliding-step residual-gradient TD was more robust than residual-gradient TD.

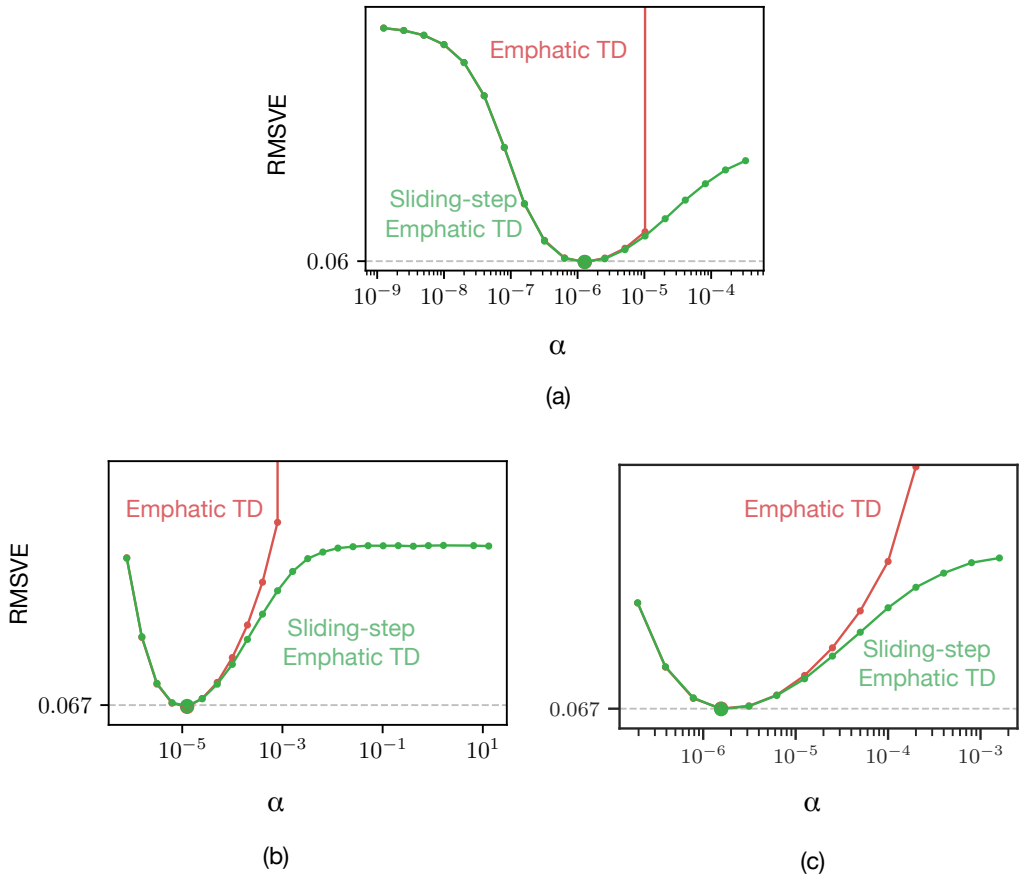

Figure 6.1: 1000-states random walk: performance comparison of sliding-step emphatic TD and emphatic TD. (a) is for 50 neighbors with tile coding. (b) and (c) are for 50 and 20 neighbors with Fourier basis. The RMSVEs are averages of 5000 episodes and then averaged over 30 runs.

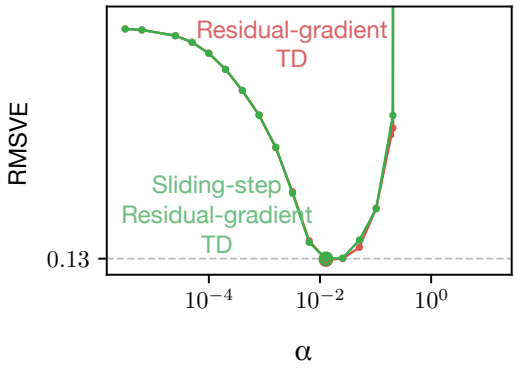

Figure 6.2: 1000-states random walk with 50 neighbors: performance comparison of sliding-step residual-gradient TD and residual-gradient TD with Fourier basis. The RMSVEs are averages of 5000 episodes and then averaged over 30 runs.

6.4 Chicken Problem

This experiment is in the off-policy setting. There are a total of 8 states with 1 terminal state. The agent starts in the first four states with equal probability. If the agent is within the first four states, then the behavior and target policy are the same, and the agent goes forward. If the agent has passed the first four states, the behavior policy chooses to go forward or go back to the first four states with equal probability. The target policy, however will always choose to go forward. If the agent goes forward and makes it to the terminal state, then it earns a reward of 1, and the episode terminates. If the agent go back to the first four states, then it receives a reward of 0 and the process continues. the discount factor is $\gamma = 0.9$.

For every run, a new set of feature vectors are randomly generated to represent the eight states. Each feature vector is $\{0, 1\}^6$, and we ensure no feature vectors are all zeros and the feature matrix is always rank 6. We ran each experiment for 5000 episodes and repeated each independently for 100 runs.

Results and Discussions

In all experiments, sliding-step algorithms performed in ranges of α that were wider than their respective TD counterparts. Sliding-step emphatic TD was more robust than emphatic TD, and sliding-step residual-gradient TD was more robust than residual-gradient TD.

Sliding-step emphatic TD with optimized $\alpha = 0.016$ learned faster than emphatic TD in the first 100 episodes as seen in figure 6.3(c). Emphatic TD outperformed sliding-step emphatic TD in the long run evident in figure 6.3(d), but the difference was small.

After optimizing for α, residual-gradient TD learned faster than sliding-step residual-gradient TD, but the latter was less noisy in the long run. We reduced the α of residual-gradient TD to 0.016 in the hope that it is less noisy in the long run, but with a smaller α, residual-gradient TD performed worse than the algorithm with $\alpha = 0.128$ (a bigger α) in the long run.

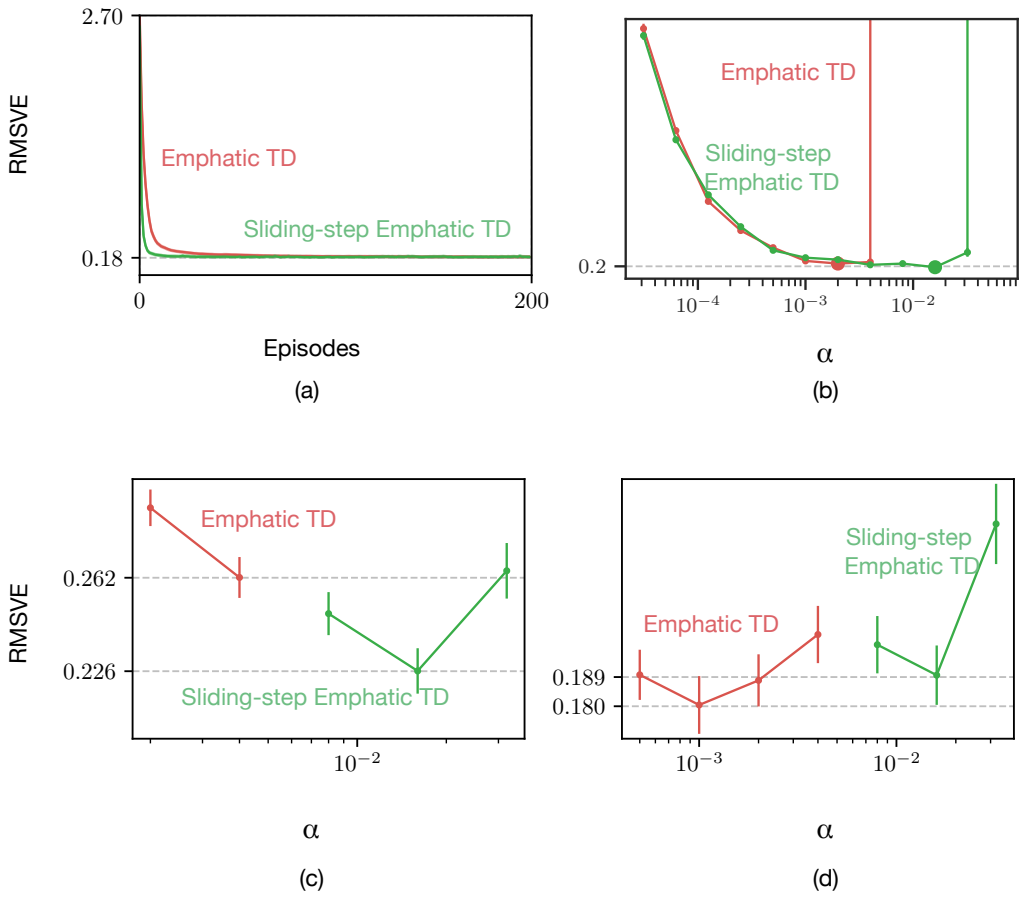

Figure 6.3: Chicken Problem: performance comparison of sliding-step emphatic TD and emphatic TD. (a) learning curves with optimized $\alpha = 0.016$ for sliding-step emphatic TD and $\alpha = 0.004$ for emphatic TD. (b) sensitivity graphs where the RMSVEs are averages of 500 episodes. (c) sensitivity graphs where the RMSVEs are averages of the first 100 episodes (early learning). (d) sensitivity graphs where the RMSVEs are averages of the last 1000 episodes (late learning). All the RMSVEs are averaged over 100 runs.

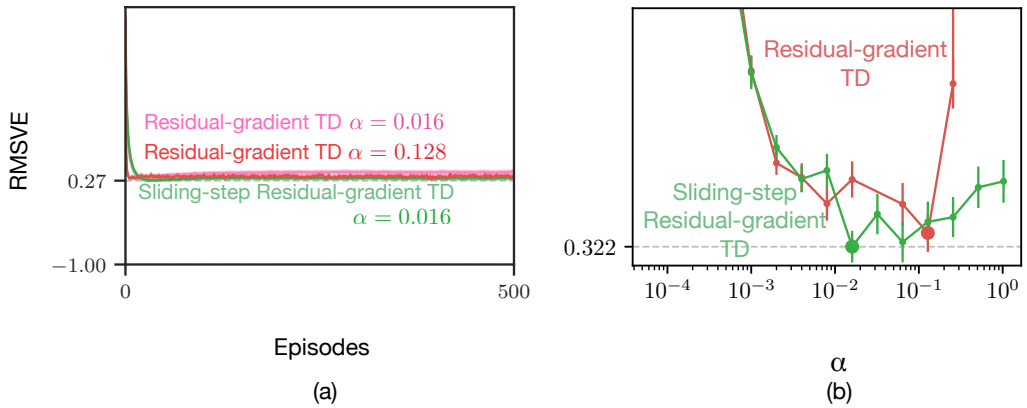

Figure 6.4: Chicken problem: performance comparison of sliding-step residual-gradient TD and residual-gradient TD. The RMSVEs for the learning curve are averaged over 100 runs. The RMSVEs for the sensitivity graph are averages of 500 episodes and then averaged over 100 runs.

Chapter 7

Conclusion

We have extended the sliding-step technique to TD learning, and presented a new algorithm called sliding-step TD. It is similar in form to TD, but differs in the expression $\frac{1-\exp(-\alpha \boldsymbol{x}^\top \boldsymbol{x})}{\boldsymbol{x}^\top \boldsymbol{x}}$ replacing the usual step-size parameter. We prove that the tabular sliding-step TD in the on-policy setting converges with probability 1. However, sliding-step TD is designed for linear function approximation, and it differs from the tabular algorithm in that it generalizes where as the tabular algorithm does not. The analysis of sliding-step TD does not follow from the tabular sliding-step TD, and we leave the formal analysis of sliding-step TD for future research.

The expression $\frac{1-\exp(-\alpha \boldsymbol{x}^\top \boldsymbol{x})}{\boldsymbol{x}^\top \boldsymbol{x}}$ within the update of the sliding-step TD depends on the state representations, or feature vectors. If the magnitude of the feature vectors are the same for all state $s \in \mathcal{S}$, then sliding-step TD deduce to TD. A few special cases involves basis unit vector and normalized vectors. Therefore, for those special cases, sliding-step have the same convergence properties as TD. For the general case, the behavior of sliding-step TD depends on the choice of α. For sufficiently small α, the expression $\frac{1-\exp(-\alpha \boldsymbol{x}^\top \boldsymbol{x})}{\boldsymbol{x}^\top \boldsymbol{x}}$ is approximately α, which means sliding-step TD performs similarly to TD. Such results have been confirmed experimentally. For an α that is not sufficiently small, the expression is different depending on the feature vectors. In this case, sliding-step TD does not average in the same way as TD, and does not converge in the neighborhood of TD.

In the empirical evaluations of the sliding-step TD, we consider two set-

tings of the parameter α: constant α and diminishing $\alpha(t)$. In all settings, we saw evidence of robustness in sliding-step TD. When the magnitude of the feature vectors varied significantly, sliding-step TD learned faster than TD. One possible explanation of the speedup could be because of the expression $\frac{1-\exp(-\alpha \boldsymbol{x}^\top \boldsymbol{x})}{\boldsymbol{x}^\top \boldsymbol{x}}$ normalizes the feature vectors, and hence bounding the size of the update to the weight vector. However, when the magnitude of the feature vectors were all large, sliding-step TD still needed a small enough α to converge, and sliding-step TD performed similarly to TD.

Lastly, we extended the sliding-step technique to emphatic TD and residual-gradient TD, which are known to have large updates to the weight vector due to emphasis and importance sampling ratios. In both the on-policy and off-policy settings, sliding-step emphatic TD was more robust than emphatic TD, and sliding-step residual-gradient TD was more robust than residual-gradient TD. In the off-policy setting, sliding-step emphatic TD learned faster than emphatic TD. To our surprise, we saw no speedup in sliding-step emphatic TD in the on-policy setting with large emphasis.

In all experiments, the sliding-step technique allowed for a wider range of α values than setting α alone. Therefore, the sliding-step technique significantly improved the robustness of the TD algorithms with respect to parameter search.

Bibliography

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Machine Learning*, pages 30–37.

Bertsekas, D. and Tsitsiklis, J. (1989). *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ.

Bertsekas, D. P. (1982). Distributed dynamic programming. *IEEE Transactions on Automatic Control*, 27:610–616.

Beygelzimer, A., Dasgupta, S., and Langford, J. (2009). Importance weighted active learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 49–56.

Borkar, V. S. (2008). *Stochastic Approximation*. Cambridge University Press, Cambridge.

Bradtke, S. J. (1994). *Incremental Dynamic Programming for On-Line Adaptive Optimal Control*. PhD thesis, University of Massachusetts Amherst.

Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57.

Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Freund, Y. and Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55:119–139.

Hinton, G., Srivastava, N., and Swersky, K. (2012). Lecture 6a overview of mini-batch gradient descent.

Karampatziakis, N. and Langford, J. (2011). Online importance weight aware updates. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, pages 392–399.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

Kushner, H. J. and Yin, G. G. (2003). *Stochastic Approximation and Recursive Algorithms and Applications*. Springer-Verlag, New York, NY.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22.

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

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

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

Sutton, R. S., Mahmood, R. A., and White, M. (2016). An emphatic approach to the problem of off-policy temporal-difference learning. *Journal of Machine Learning Research 17*, 73:1–29.

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16:185–202.

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

Yu, H. (2015). On convergence of emphatic temporal-difference learning. arXiv preprint arXiv:1506.02582; a shorter version appeared in the 28th Annual Conference on Learning Theory (COLT) 2015.