Online Off-policy Prediction

by

Sina Ghiassian

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

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

In this dissertation, we study online off-policy temporal-difference learning algorithms, a class of reinforcement learning algorithms that can learn predictions in an efficient and scalable manner. The contributions of this dissertation are one of the two kinds: (1) empirically studying existing off-policy learning algorithms, or, (2) exploring new algorithmic ideas.

In reinforcement learning, it is not uncommon for an agent to learn about one policy, called the target policy, while behaving using a different policy, called the behavior policy. When the target and behavior policies are different, learning is 'off' the policy in the sense that the data is from a different source than the target policy.

Our first contribution is a novel Gradient-TD algorithm called TDRC. Gradient-TD algorithms are one of the most important families of off-policy learning algorithms due to their favorable convergence guarantees. Previous research showed that Gradient-TD algorithms learn slower than some other unsound algorithms such as Off-policy TD(λ) (Maei, 2011). In addition, Gradient-TD algorithms are not as easy to use as Off-policy TD because they have two learned weight vectors and two step-size parameters. Our contribution is a new algorithm called temporal-difference learning with regularized corrections (TDRC), that behaves as well as Off-policy TD, when Off-policy TD performs well but is sound in cases where Off-policy TD diverges. TDRC provides a standard way of setting the second step-size parameter and is easier to use than other Gradient-TD algorithms. We empirically investigate the performance of TDRC across a range of problems, for both prediction and control, and for both linear and non-linear function approximation, and show that it consistently outperforms other Gradient-TD algorithms. We derive

the control variant of TDRC and show that it could be a better alternative to existing algorithms for solving complex control problems with neural networks.

One of the most important contributions of this thesis is a comprehensive empirical study of off-policy prediction learning algorithms. The study is one of the most thorough studies of off-policy prediction learning algorithms to date in terms of the number of algorithms it includes and is unique in that the performance of algorithms with respect to each of the algorithms' parameters is assessed individually. We present empirical results with eleven prominent off-policy learning algorithms that use linear function approximation: five Gradient-TD methods, two Emphatic-TD methods, Off-policy TD(λ), V-trace(λ), Tree Backup(λ), and ABQ(ζ). We assess the performance of the algorithms in terms of their learning speed, asymptotic error, and sensitivity to the step-size and bootstrapping parameters on a simple problem, called the Collision task, that has eight states and two actions. On the Collision task, we found that the two Emphatic-TD algorithms learned the fastest, reached the lowest errors, and were robust to parameter settings. V-trace, Tree Backup, and ABQ were no faster and had higher asymptotic error than other algorithms.

Another one of the main contributions of this thesis is the first empirical study of off-policy prediction learning algorithms with a focus on one of the most important challenges in off-policy learning: slow learning. When learning off-policy, importance sampling ratios are used to correct for the differences between the target and behavior policies. These ratios can be large and of high variance. The step-size parameter should be set small to control this high variance, which in turn slows down learning. In this study, we found that the algorithms' performance is highly affected by the variance induced by the importance sampling ratios. Data shows that algorithms that adapt the bootstrapping parameter during learning, such as V-trace(λ) are less affected by the high variance than the other algorithms. We observed that Emphatic TD(λ) tends to have lower asymptotic error than other algorithms, but exhibits high variance and does not learn well when the product of importance sampling ratios is large.

The final contribution of this thesis is a step-size adaptation algorithm, called Step-size Ratchet (and a variant of it called Soft Step-size Ratchet), that when combined with Emphatic TD(λ), significantly improves its learning speed. The Step-size Ratchet algorithm keeps the step-size parameter as large as possible and only ratchets it down when necessary. We establish the effectiveness of the combination of Emphatic TD and Step-size Ratchet by comparing it with the combination of Emphatic TD(λ) and other step-size adaptation algorithms on tasks where learning fast is challenging.

The empirical studies conducted in this dissertation are not the last word, but they contribute to the growing database of reliable results comparing modern off-policy learning algorithms. Similarly, the algorithms proposed in this dissertation do not completely resolve the challenges of off-policy prediction learning, but they do take a step in increasing the practicality of the existing set of off-policy prediction learning algorithms.

Preface

Several parts of this dissertation are based on papers that were written in collaboration with others.

Chapters 4 and 5 are based on a published paper that introduces the TDRC algorithm, a new variant of Gradient-TD algorithms (Ghiassian, Patterson, Garg, Gupta, White, & White 2020). This work was motivated by the preformance gap we observed between the sound GTD(λ) algorithm, and the unsound Off-policy TD(λ) algorithm. I was involved in all parts of the work, including the algorithmic, experimental, theoretical, and writeup aspects. Other authors were also involved in pretty much all aspects of the work. Shivam Garg, specifically played a key role in establishing that a variant of the proof provided by Maei (2011) for the convergence of the TDC algorithm applies to the TDRC algorithm.

Chapter 6 is based on a paper under review (Ghiassian & Sutton, 2021a), mostly done by Sutton and I. I was responsible for designing, conducting, and writing about the experiments and the results. I discussed extensively every aspect of the project with Sutton during the course of the write up. Sutton had a key role in forming the architecture of the manuscript. An earlier version of this paper is also availabble on ArXiv. Adam White, Martha White, and Andrew Patterson helped me in writing up an earlier version of the paper. Adam White and Martha White's help for the earlier version of the paper included but was not limited to forming the overall architecture of the paper.

Chapter 7 is also based on a paper under review (Ghiassian & Sutton, 2021b), mostly done by Sutton and I. I was responsible for designing and conducting the experiments. I was also responsible for writing up the manuscript. Sutton helped me to a great extent in presenting the work and editing the manuscript. Similar to the previous chapter, Adam White, Martha White, and Andrew Patterson helped me in writig up the earlier version of the paper. Adam White and Martha White's help on the early version of the paper included but was not limited to forming the overall architecture of the paper. Kenny Young helped me in preparing Section 7.11.

Chapter 8 is based on a paper in preparation. Sutton and I developed the algorithm and designed the experiments. I was responsible for conducting the experiments, analyzing the results and writing up the work.

To my mother and father

One day I will find the right words, and they will be simple.

– Jack Kerouac.

Acknowledgements

This thesis is the product of my collaboration with many great scientists. First and foremost I would like to express my deepest appreciation to my supervisor, Richard Sutton, without whom, this thesis would not have been possible. Rich is a deep thinker, a visionary scientist, and has a clear and unique view of the Artificial Intelligence field. He spent an enormous amount of time teaching me how to approach problems, how to think about solving them, and finally how to communicate the results and conclusions to other scientists. At times during my PhD, we met for several hours, five days a week, in spite of his extremely tight schedule. Those times definitely were the times that I made the greatest progress in my program. Rich has definitely been an excellent mentor and a good friend. It has been an honor to work alongside him during the course of my PhD studies.

I want to thank Adam White and Martha White. They joined the University of Alberta when I was in the third year of my PhD, and fueled my research to a great extent since then. Adam involved himself in all and every part of a research project, and was helpful at many different levels, from forming the high level idea to the write up. Martha's door was always open to students. I am grateful to Dale Schuurmans and Michael Bowling who allowed me to check with them from time to time to discuss my progress and provided useful feedback. I am thankful to Peter Stone, for useful discussion during the defence, and for agreeing to serve as a member of my final examination committee.

I want to thank Banafsheh Rafiee, who supported me mentally and intellectually at each and every step of this thesis. She was essential in getting through the hard times.

I owe an important debt to Ali Khalili, who spent countless hours with me working on cleaning up the source codes that I used to produce many of the results of this thesis.

I owe many thanks to my father, mother, and sister, who supported me with their endless encouragement from the very beginning. This thesis would not be possible without them.

I owe many thanks to my friends and lab-mates, Dylan Ashley, Shivam Garg, Andrew Jacobsen, Raksha Kumaraswamy, Payam Mousavi, Arsalan Sharifnassab, and Kenny Young, who helped in proof reading the dissertation.

I would like to thank my co-authors and collaborators Shivam Garg, Dhawal Gupta, and Andrew Patterson. I am also thankful to the undergraduate students who helped me gain valuable mentoring experience: Yat Long Lo and Xiang Gu. Last but not least, I want to thank my friends, Zaheer Abbas, Mohammad Ajallooeian, Kavosh Asadi, Eric Graves, Khurram Javed, Derek Li, Pooria Joulani, Soheil Kianzad, Abhishek Naik, Bradley Poulette, Arash Tavakoli, Han Wang, Yi Wan, and other members of the RLAI lab.

Special thanks goes to DeepMind, the Alberta Machine Intelligence Institute (Amii), Natural Sciences and Engineering Research Council of Canada (NSERC), Canadian Institute for Advanced Research (CIFAR), JPMorgan Chase & Co., and the University of Alberta for funding my PhD studies.

Contents

List of Figures

Chapter 1

Learning Predictions in a Continual, Scalable, and Online Manner

We make hundreds, if not thousands of decisions every day. In every decision, one or more predictions are involved. These predictions form our expectation of what happens next if an action is taken in a situation. For example, when we decide to open a door, we make a number of decisions and make predictions about each of them: we reach for the door knob, predicting that our hand will touch the door knob, turn it, predicting that the knob will in fact turn and predicting that the door will be unlocked, and finally pull the door predicting that the door will be opened. The collection of these predictions that we have about our behavior constitutes much of our *knowledge* about interacting with the world. In this sense, much of *knowledge is predictive*.

To learn predictions that constitute predictive knowledge, like the ones mentioned in the example above, it is best that the learning algorithm has special attributes. First, it is desirable if the algorithm is able to learn in a *continual* manner. Learning should continue forever, because we want the agent to be able to adapt to new situations and learn how to deal with them. Second, it is desirable if the algorithm is able to learn in a *scalable* manner because there are many predictions to be learned about and the agent should be able to learn and maintain a large body of predictive knowledge. The agent and the world are both complex, and the intelligent agent needs scalable

algorithms to be able to handle the complexity. Finally, it is desirable if the algorithm is able to learn in an *online* manner, meaning that the agent learns as it interacts with its environment. This is in contrast to storing the data and learning from it later on. We want learning algorithms to be able to learn online because it can be more robust than offline learning, meaning that errors in the data can be corrected during the operation, whereas in offline learning, the data is gathered and fixed, and possible errors cannot be corrected while the agent is learning from the data. Another reason that we want learning to be online is that the data for online learning can often be generated at low cost and in large quantities. For example, AlphaGo, the computer Go player made by Silver et al. (2016), that beat the human champion, or the success of Tesauro's (1994) TD Gammon both rely on a vast amount of data generated by an agent while in operation. Finally, to adapt to changes in the environment, online learning is important as is learning continually (Sutton & Whitehead, 1993).

1.1 Off-policy Prediction Learning

Off-policy prediction learning algorithms, which are the focus of this dissertation, can be utilized by an intelligent agent to learn predictions in a continual and scalable manner. In reinforcement learning, it is not uncommon to learn a value function for a policy while following another policy. For example, the Q-learning algorithm (Watkins, 1989; Watkins & Dayan, 1992) learns the value of the greedy policy while the agent selects its actions according to a different, more exploratory policy. The first policy, the one whose value function is being learned, is called the *target policy* while the more exploratory policy generating the data is called the *behavior policy*. When the two policies are different, as they are in Q-learning, the problem is said to be one of *off-policy learning*, whereas if they are the same, the problem is said to be one of *on-policy learning*. The former is 'off' in the sense that the data is from a different source than the target policy, whereas the latter is from data that is 'on' the policy. Off-policy learning is more difficult than on-policy learning and subsumes it

as a special case.

One reason for interest in off-policy learning algorithms is that they provide a way of intermixing exploration and exploitation. The classic dilemma is that an agent should always *exploit* what it has learned so far—it should take the best actions according to what it has learned—but, on the other hand, it should always *explore* to find actions that might be superior. Of course, no agent can simultaneously behave in both ways. However, an off-policy algorithm like Q-learning can, in a sense, pursue both goals at the same time. The behavior policy can explore freely, while the target policy can converge to the fully exploitative, optimal policy independent of the behavior policy's explorations.

Another appealing aspect of off-policy learning is that it enables learning about many policies in parallel. Once the target policy is freed from behavior, there is no reason to have a single target policy. With off-policy learning, an agent could simultaneously learn how to optimally perform many different tasks (as suggested by Jaderberg et al., 2016 and Rafiee et al., 2019). Parallel off-policy learning of value functions has even been proposed as a way of learning general, policy-dependent, world knowledge (e.g., Sutton et al., 2011; White, 2015; Ring, in prep).

Finally, note that numerous ideas in the machine learning literature rely on effective off-policy learning, including the learning of temporally-abstract world models (Sutton, Precup, & Singh, 1999), predictive representations of state (Littman, Sutton, & Singh, 2002; Tanner & Sutton, 2005), auxiliary tasks (Jaderberg et al., 2016), life-long learning (White, 2015), and learning from historical data (Thomas, 2015).

Many off-policy learning algorithms have been explored in the history of reinforcement learning. Q-learning (Watkins, 1989; Watkins & Dayan, 1992) is perhaps the oldest. In the 1990s it was realized that combining off-policy learning, function approximation, and temporal-difference (TD) learning risked instability (Baird, 1995). Precup, Sutton, and Singh (2000) introduced off-policy algorithms with importance sampling and eligibility traces, as well as tree backup algorithms, but did not provide a practical solution to the risk of instability. Gradient-TD methods (see Maei, 2011; Sutton et al., 2009) as-

sured stability by following the gradient of an objective function, as suggested by Baird (1999). Emphatic-TD methods (Sutton, Mahmood, & White, 2016) reweighted updates in such a way as to regain the convergence assurances of the original on-policy TD algorithms. These methods had convergence guarantees, but no assurances that they would be efficient in practice. Other off-policy algorithms, including Retrace (Munos et al., 2016), V-trace (Espeholt et al., 2018), and ABQ (Mahmood, Yu, & Sutton, 2017) were developed recently to overcome difficulties encountered in practice.

As more off-policy prediction learning algorithms were developed, there was a need to compare them systematically. There have been a few systematic empirical studies of off-policy prediction learning algorithms. The earliest systematic study was probably that by Geist and Scherrer (2014). Their experiments were on random MDPs and compared eight major off-policy algorithms. A few months later, Dann, Neumann, and Peters (2014) published a more in-depth study with one additional algorithm (an early Gradient-TD algorithm) and six test problems including random MDPs. Both studies considered off-policy problems in which the target and behavior policies were given and stationary. Such *prediction* problems allow for relatively simple experiments and are still challenging (e.g., they involve the same risk of instability). Both studies also used linear function approximation with a given feature representation. The algorithms studied by Geist and Scherrer (2014), and by Dann, Neumann, and Peters (2014) can be divided into those whose per-step complexity is linear in the number of parameters, like TD(λ), and methods whose complexity is quadratic in the number of parameters (proportional to the square of the number of parameters), like Least-squares TD(λ) (Bradtke & Barto, 1996; Boyan, 1999). Quadratic-complexity methods avoid the risk of instability, but cannot be used in learning systems with large numbers (e.g., millions) of weights. A third systematic study, by White and White (2016), limited the breadth of the study in that they excluded quadratic-complexity algorithms, but added four additional linear-complexity algorithms and additionally, studied different forms of eligibility traces.

Although there has been a few studies that focused on practical considera-

tions in off-policy prediction learning, there remains a gap between the theory of off-policy learning and its practical applications as most of the recent work in off-policy learning has focused on guaranteeing stability with linear function approximation and left a thorough empirical evaluation of algorithms for future work.

1.2 Objective

This dissertation seeks to answer the following question:
How do online off-policy prediction learning algorithms compare to each other in practice and how can their practicality be improved?
At this point, the reader might rightfully ask what practicality means. We generally examine three areas when studying a prediction algorithms practicality. One important area is learning speed. It is important that the algorithm learns fast, meaning that it is able to learn predictions via minimal interactions with the environment. Other than being fast, we care about the final error level. An algorithm that converges fast but to a poor solution is not desirable. Finally, when studying the performance of algorithms with respect to an error measure, the performance changes as we adjust the algorithms' parameters. The less we need to adjust the parameters of an algorithm in order for it to learn fast and accurately, the better.

We empirically investigate the practicality of prominent prediction learning algorithms by applying them to three small problems of increasing size and complexity. Sensitivity curves are the main tool we use for assessing the algorithms' practicality. Sensitivity curves show the performance of an algorithm over many parameter settings in a condensed manner. In sensitivity curves, the primary measure of good performance that we look for is low error, over a large range of parameters. This measure can tell us if the algorithm learned a task well, and how easy it was to tune the algorithm, depending on what portion of parameters resulted in low error.

To improve algorithms' practicality, our methodology is to closely examine the existing algorithms, spot their weaknesses and make improvements where

they have a weakness. When examining the practicality of existing algorithms, it is observed repeatedly that an algorithm performs better than another in one aspect but performs worse in another aspect. The latter is referred to as a weakness of the algorithm. For example, the Gradient-TD algorithms have stronger convergence guarantees than Off-policy TD(λ) but they learn slower. One of the objectives of this dissertation is to make incremental improvements to algorithms where they exhibit a weakness.

1.3 Contributions

This dissertation makes five main contributions, including two empirical studies and three algorithmic ideas.

The two empirical studies go deeper than other empirical studies of off-policy prediction learning algorithms conducted to date in that they include more online off-policy prediction learning algorithms than any other study conducted to date. Additionally, the empirical studies in this dissertation in a sense go deeper than the previous empirical studies in that they analyze the performance of algorithms with respect to the parameters of algorithms individually rather than maximizing over one parameter to study the performance over another. This provides a more realistic picture of the performance of algorithms and can result in a better understanding of the algorithms. Our empirical studies are more limited than the previous empirical studies in breadth, in that they only consider linear complexity algorithms and focus only on practical considerations. Our empirical studies are more limited than the one conducted by White and White (2016) in that only one form of the eligibility trace is studied. Last but not least, our empirical studies are more limited than the previous studies in that they do not consider random MDPs, whereas all previous studies of off-policy prediction learning algorithms included some form of random MDPs in their experiments.

The three algorithmic ideas have their root in empirical analyses, in that their goal is to improve the algorithms where they showed to have a weakness in empirical studies. One of the main algorithmic ideas for prediction

learning uses regularization techniques and is motivated by the performance gap observed by Maei (2011) between Off-policy TD(λ) and Gradient-TD algorithms. In supervised learning, regularization is used to encourage learning a simple model. Learning simple models can be useful, especially when we do not want to overfit to possible noise in the data. Gradient-TD algorithms use a correction term in their update rule so that the updates are in the direction of minimizing an objective function. This correction term relies on a learned weight vector that we call the secondary weight vector. Secondary weights are learned such that they linearly approximate the TD-error. Because the TD-error can itself be noisy, using regularization can help learn better approximations of the TD-error. The second algorithmic idea is a simple modification of the first one so that it is applicable to control. The third algorithmic idea is a step-size adaptation algorithm to increase the learning speed of Emphatic TD(λ). The proposed algorithm, called Step-size Ratchet, shrinks down the step-size parameter over the course of learning only when it is necessary for the step-size parameter to be shrunk to avoid overshoot. Below, we discuss the contributions of this dissertation in more detail.

The TDRC algorithm (Chapter 4)

We propose a new algorithm called TD with Regularized Corrections (TDRC), that attempts to balance ease of use, soundness, and performance. Maei (2011) showed that although Gradient-TD algorithms have strong convergence properties, they learn slower than the classic Off-policy TD(λ) algorithm that is not guaranteed to converge under off-policy training. Our new algorithm, TDRC(λ) uses regularization of the weight vector, to improve the learning speed and preserve convergence properties. We show that TDRC(λ) behaves as well as Off-policy TD(λ), when Off-policy TD(λ) performs well, but is sound in cases where Off-policy TD(λ) diverges. We empirically investigate TDRC(λ) across a range of problems for both prediction with linear function approximation and show that it outperforms other Gradient-TD algorithms.

The QRC Algorithm (Chapter 5)

We extend TDRC such that it is applicable to control problems. The resulting algorithm is called Q-learning with Regularized Corrections (QRC). We show that QRC performs as well as the classic Q-learning algorithm when applied to simple tasks such as Mountain Car, and also show that it outperforms the DQN architecture (Mnih et al., 2015) on two challenging tasks where an agent learns how to play games from raw pixels.

An empirical comparison of online off-policy prediction learning algorithms on the Collision task (Chapter 6)

We present empirical results with eleven prominent off-policy prediction learning algorithms with linear function approximation: five Gradient-TD methods, two Emphatic-TD methods, Off-policy TD(λ), V-trace, and variants of Tree Backup and ABQ derived in this dissertation so that they are applicable to the prediction setting. Our experiments used the Collision task, a small off-policy problem analogous to that of an autonomous car trying to predict whether it will collide with an obstacle. We assessed the performance of the algorithms according to their learning rate, asymptotic error level, and sensitivity to step-size and bootstrapping parameters. By these measures, the eleven algorithms can be partially ordered on the Collision task. In the top tier, the two Emphatic-TD algorithms learned the fastest, reached the lowest errors, and were robust to parameter settings. In the middle tier, the five Gradient-TD algorithms and Off-policy TD(λ) were more sensitive to the bootstrapping parameter. The bottom tier comprised V-trace, Tree Backup, and ABQ; these algorithms were no faster and had higher asymptotic error than the others. Our results are definitive for this task, though of course experiments with more tasks are needed before an overall assessment of the algorithms' merits can be made.

An empirical comparison of online off-policy prediction learning algorithms in the Four Rooms environment (Chapter 7)

Many off-policy prediction learning algorithms have been proposed in the past

decade, but it remains unclear which algorithms learn faster than others. We empirically compare the same set of off-policy prediction learning algorithms mentioned above with linear function approximation on two small tasks: the Rooms task, and the High Variance Rooms task. The tasks are designed such that learning fast in them is challenging. In the Rooms task, the product of importance sampling ratios can be as large as 2^{14}. To control the high variance caused by the product of the importance sampling ratios, the step-size parameter should be set small, which in turn slows down learning. The High Variance Rooms task is more extreme in that the product of the ratios can become as large as $2^{14} \times 25$. We consider the same set of algorithms as in the Collision task and employ the same experimental methodology. The algorithms considered are: Off-policy TD(λ), five Gradient-TD algorithms, two Emphatic-TD algorithms, Tree Backup(λ), V-trace(λ), and ABTD(ζ). We found that the algorithms' performance is highly affected by the variance induced by the importance sampling ratios. The data shows that Tree Backup(λ), V-trace(λ), and ABTD(ζ) are not affected by the high variance as much as other algorithms but they restrict the effective bootstrapping parameter in a way that is too limiting for tasks where high variance is not present. We observed that Emphatic TD(λ) tends to have lower asymptotic error than other algorithms, but might learn more slowly in some cases.

The Step-size Ratchet algorithm (Chapter 8)

We propose a new algorithm for adapting the step-size parameter of Emphatic TD(λ). We observed in our empirical studies that Emphatic TD(λ) might learn slower than other algorithms when solving a problem with large products of importance sampling ratios. The Step-size Ratchet algorithm ratchets down the step-size when necessary to avoid overshoot. We show that Step-size Ratchet significantly improves Emphatic TD(λ)'s learning speed. We show that the combination of Emphatic TD(λ) and Step-size Ratchet is perfectly capable of solving problems that Emphatic TD(λ) with constant step-size parameters has trouble solving due to high variance and slow learning. We follow up with a new form of Step-size Ratchet which we call Soft Step-size Ratchet, that

occasionally increases the magnitudes of the step-size parameter to provide even further improvements in learning speed. We show that on the Rooms, and the High Variance Rooms tasks, Soft Step-size Ratchet combined with Emphatic TD(λ) outperforms Emphatic TD(λ) combined with other step-size adaptation algorithms such as Adam.

1.4 Summary

In this chapter, we discussed the main topic of this dissertation, the objective, and some of the key ideas that are used throughout the thesis. We went over some of the related works, and finally closed by briefly discussing the contributions that we make throughout the dissertation.

Chapter 2

Background

This thesis includes two empirical studies and three novel algorithmic ideas for online off-policy prediction learning. To understand the contributions of this dissertation, an understanding of a few fundamental reinforcement learning concepts including value functions, prediction learning, off-policy learning, and function approximation is needed. The contents of this chapter serve as background only. The reader familiar with the concepts mentioned above, can safely skip this chapter.

We start by going over the reinforcement learning framework and Markov Decision Processes. We continue with a formal definition of state-value functions and state-action-value functions, and finish with a formal discussion of off-policy learning, importance sampling ratios, and linear and non-linear function approximation.

2.1 Reinforcement Learning

In reinforcement learning, an agent and environment interact at discrete time steps, $t = 0, 1, 2, \ldots$. The environment is a Markov Decision Process (MDP) and is in state $S_t \in \mathcal{S}$ at time step t. At each time step, the agent chooses an action $A_t \in \mathcal{A}$ with probability $\pi(a|s)$, where the function $\pi : \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ with $\sum_{a \in \mathcal{A}} \pi(a|s) = 1, \forall s \in \mathcal{S}$, is called the policy and determines the agent's behavior. After taking action A_t in state S_t, the agent receives from the environment a numerical reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and the environment moves to a new state S_{t+1}. The reward and the next state are stochastically

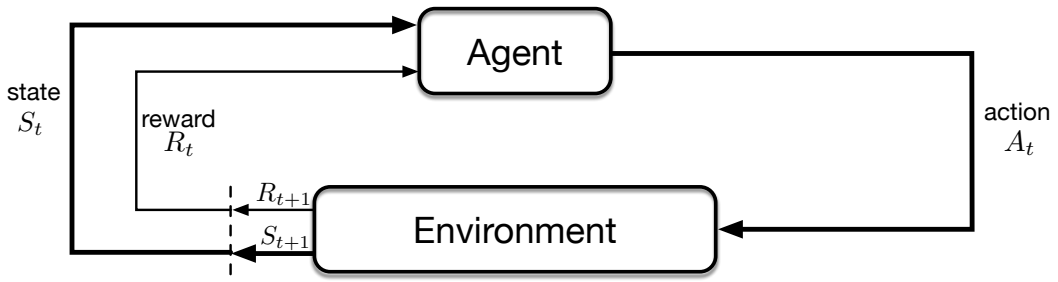


Figure 2.1: The agent and environment interact with each other at discrete time steps. At each time step, the environment is in a state S_t, the agent takes an action A_t, and as a result receives a numerical reward R_{t+1} and the environment moves to a next state S_{t+1}.

jointly determined using the MDP's transition dynamics of the environment: $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{R} \to [0, 1]$, such that $p(s', r|s, a) \stackrel{\text{def}}{=} Pr(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$ for $\forall s, s' \in \mathcal{S}, a \in \mathcal{A},$ and $r \in \mathcal{R}$. See Figure 2.1.

The discounted sum of future rewards at time t is called the *return* and is denoted by G_t:

$$G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots,$$

where γ is called the *discount factor*. The expected return when starting from a state and following a specific policy thereafter is called the *value* of the state under the policy. The *state-value function*, or for short, the *value function* $v_\pi : \mathcal{S} \to \mathbb{R}$ for a policy π takes a state as input and returns the value of that state:

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}[G_t \mid S_t = s, A_{t:\infty} \sim \pi]. \tag{2.1}$$

Termination functions are a generalization of discount factors (White, 2017). The termination function is a function of the current state, the current action, and the next state: $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. With termination functions instead of the discount factor, the return is defined as:

$$G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma(S_t, A_t, S_{t+1})R_{t+2} + \gamma(S_t, A_t, S_{t+1})\gamma(S_{t+1}, A_{t+1}, S_{t+2})R_{t+3} + \cdots$$

If for some triplet, (S_k, A_k, S_{k+1}), the termination function returns zero, the accumulation of the rewards will be terminated after time step k without affecting the behavior trajectory of the agent.

The predictions that we discussed in the first chapter can be formulated through value functions. In prediction learning, the goal of the agent is to learn value functions through interaction with the environment.

In control learning, it is common to use the *state-action-value function for policy* π or simply the *action-value function*. The action-value function for policy π is defined as:

$$q_\pi(s,a) \stackrel{\text{def}}{=} \mathbb{E}\left[G_t \mid S_t = s, A_t = a, A_{t+1:\infty} \sim \pi\right]. \tag{2.2}$$

The main difference between state-value functions and action-value functions is that the expectation is computed given the state and the action for action-value functions, while for state value functions, the expectation is computed given the state only.

In control learning, the goal typically is to learn the policy whose action-value function assigns to each state-action pair the largest expected return achievable by any policy. This policy is called the optimal policy:

$$q_*(s,a) \stackrel{\text{def}}{=} \max_\pi q_\pi(s,a),$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$.

There might sometimes be natural breaks in the interaction of the agent-environment. We call the intervals between these breaks *episodes*. Each episode ends in a state called the *terminal state*. Reaching the terminal state is typically followed by a reset to the starting state or to a sample from the distribution of starting states. Reinforcement learning tasks that fit in the above-mentioned explanation, are called *episodic tasks*. In contrast, in many cases, learning continues forever without limit. Reinforcement learning tasks of this kind are called *continuing tasks*. Notation for episodic and continuing tasks can be unified by considering an *absorbing state*, that transitions only to itself with a reward of zero. In this dissertation, we study both episodic and continuing tasks.

2.2 Off-policy Learning

In off-policy learning, the agent follows a *behavior policy* b, and is interested in learning the value function under a different target policy, π. To account for the differences between the target and the behavior policies, importance sampling ratios are typically used. The importance sampling ratio is the probability of taking an action in a state under the target policy divided by the same probability under the behavior policy. Formally, the importance sampling ratio is defined as:

$$\rho_t \stackrel{\text{def}}{=} \frac{\pi(A_t|S_t)}{b(A_t|S_t)}, \tag{2.3}$$

which has an expected value of one:

$$\mathbb{E}_b[\rho_t \mid S_t = s] = \sum_a b(a|s)\frac{\pi(a|s)}{b(a|s)} = \sum_a \pi(a|s) = 1, \tag{2.4}$$

where $\mathbb{E}_b[\cdot \mid \cdot]$ is the conditional expectation given than policy b is followed. At time steps where the target and behavior policies have the same distribution and in on-policy learning, the ratio is one. When the target and behavior policies are different, the ratio will be greater or less than one, depending on if the action that was taken under the behavior policy would have been more or less likely under the target policy.

For any random variable X_{t+1} that is generated using the behavior policy, and depends on the state-action-next state triplet, the expectation under the target policy can be computed using importance sampling ratios:

$$\begin{aligned}
\mathbb{E}_b[\rho_t X_{t+1} \mid S_t = s] &= \sum_a b(a|s)\frac{\pi(a|s)}{b(a|s)}X_{t+1} \\
&= \sum_a \pi(a|s)X_{t+1} \\
&= \mathbb{E}_\pi[X_{t+1} \mid S_t = s]\,, \qquad \forall s \in \mathcal{S}.
\end{aligned}$$

2.3 Linear and Non-linear Function Approximation

In the simplest form, the agent observes the state, and estimates a value function for each of the observed states separately. This setting is known as

the *tabular setting*, because the values of the states can simply be stored in a lookup table.

In most problems $\mathcal{S}$ is large and an exact approximation is not possible even in the limit of infinite time and data. This means that it is not possible to store one value for each state and the agent observes a representation of the state, rather than observing the state itself. In such cases, typically parametric function approximation is used to approximate the value function. Many parametric forms are possible, but of particular interest, and our exclusive focus in this dissertation, is the linear form. Prediction learning algorithms seek to learn an estimate $\hat{v} : \mathcal{S} \to \mathbb{R}$ that approximates the true value function v_π. In the linear form, learning algorithms seek to find a weight vector, $\mathbf{w}$, such that the product of the weight vector and the feature representation approximates the value function:

$$\hat{v}(s, \mathbf{w}) \stackrel{\text{def}}{=} \mathbf{w}^\top \mathbf{x}(s), \tag{2.5}$$

where $\mathbf{w} \in \mathbb{R}^d$ is a learned weight vector and $\mathbf{x}(s) \in \mathbb{R}^d, \forall s \in \mathcal{S}$ is a set of given feature vectors, one per state, where $d \ll |\mathcal{S}|$. In control learning, it is common for the feature vector to represent the state-action pair, instead of the state alone, in which case $\mathbf{x}(s, a)$ is used instead of $\mathbf{x}(s)$. In this case, the action-value function, $\hat{q}(s, a)$ is defined as the dot product of the weight vector $\mathbf{w}$ and the feature representation of the state action pair, (s,a):

$$\hat{q}(s, a, \mathbf{w}) \stackrel{\text{def}}{=} \mathbf{w}^\top \mathbf{x}(s, a).$$

See Chapter 9 of Sutton and Barto (2018), for a detailed explanation of linear function approximation.

Non-linear function approximation is also possible. Approximating value functions non-linearly comes with its own advantages and disadvantages. For example, non-linear functions can approximate more complex functions, but are typically learned slower and they are harder to interpret and understand.

Artificial neural networks have been widely successful in reinforcement learning when used for non-linear function approximation. An artificial neural network is a combination of nodes and real-valued weights that connect the nodes. Typically, one or more real-valued inputs are fed to the neural

network, and then each node computes a weighted sum of its inputs, and passes the weighted sum through a non-linearity, such as the logistic function, and sends the result to the next layer. The simplest form of artificial neural networks are feedforward networks. Feedforward networks do not have any loops in their connections, meaning that none of the outputs of the nodes can influence the input to the same node.

The backpropagation algorithm is a key algorithm to adjust the weights in a neural network such that the output of the network approximates a desired function. After each forward pass in the network, the partial derivative of an objective function with respect to each of the weights is calculated, and the weights are then adjusted using stochastic gradient descent. When combined with reinforcement learning algorithms, the weights of the artificial neural network are updated using reinforcement learning algorithms such as semi-gradient Q-learning or semi-gradient TD(λ). Other than the experiments of Chapter 5, the rest of the experiments in this dissertation use linear function approximation.

2.4 Online and Offline Learning

Remember that the focus of this dissertation is on online fully-incremental learning algorithms. We say that learning is online, if interaction with the environment and learning from samples received from the environment happen simultaneously. In this dissertation, we distinguish between two kinds of online learning. *Online fully-incremental learning* is the setting in which the agent receives samples from the environment one by one, learns from each single sample once it is received, and then disposes of the sample. In the other setting, which we refer to as *online weakly-incremental learning*, the agent has some kind of a memory that it uses to store samples. In the weakly-incremental learning setting, the agent receives and learns from samples once in a while. In such a setting, the agent does not wait until it receives all the data before it starts learning, nor does it necessarily update its parameter after receiving each sample. In this setting, the agent might learn from mini-batches of data.

An example of such a setting is the popular DQN architecture (Mnih et al., 2015), in which experience replay buffers are used to store data, so that they can be presented to the agent repeatedly as mini-batches, from which the agent learns a policy. Throughout this dissertation, we use the phrase "online learning", to refer to the online fully-incremental setting.

Online learning is of course not the only type of learning. For example, many supervised learning algorithms learn *offline*, meaning that the samples are all provided to the agent first, in pairs, from which the agent learns a function that maps the first element of the pair to the second one. We refer to a setting as *offline reinforcement learning* setting if the whole set of data is first collected and is then presented to the agent. The agent can for example use the dataset to learn a function that maps states to value functions. An example of offline learning is to learn from historical data. The data is gathered in the past, and is now used to learn a value function, or a policy.

Online and offline learning have their advantages and disadvantages. One advantage of online learning is that errors or omissions in the training data can be corrected during operation. For example, when a robot's sensor is not producing accurate signals, the agent might be able to omit some of the noise by averaging over sensor readings. Another advantage of online learning is that, data generation can be easy and thus great quantities of data can be generated. This is for example true in simulated problems. Yet another advantage of online learning is that the algorithms that are able to learn online can be deployed in non-stationary environments. In such environments, online learning enables the agent to adapt to a change (see Sutton & Whitehead, 1993). Of course, offline learning has its own advantages. For example, many convergence proofs rely on using independent samples for updates. In online fully-incremental learning, it is not possible to use independent samples to update the weight vector, because the samples that are received are correlated and the agent needs to learn from the samples one by one as they are received.

The focus of this dissertation is on fully-incremental online learning algorithms. These algorithms are discussed in detail in the next chapter. Nevertheless, there are many algorithms proposed for offline off-policy learning. For

example, the distribution correction estimation (DICE) family of algorithms correct for the difference in the stationary distribution of the behavior and target policies (Nachum et al., 2019a; Nachum et al., 2019b; Zhang et al., 2019). To make such corrections, lots of data is necessary, which means these algorithms are not readily available to be used in the online fully-incremental setting. A study of these algorithms is out of the scope of this dissertation.

2.5 Summary

In this chapter, we provided a review of the fundamental concepts in reinforcement learning. The concepts we explained included linear and non-linear function approximation, value functions, off-policy learning, online learning, and offline learning.

Chapter 3

Learning Algorithms for Solving Off-policy Learning Problems

Two of the most important contributions of this dissertation are empirical studies of fully-incremental off-policy prediction learning algorithms. In the empirical studies, we are interested in a wide variety of algorithms. This chapter explains these algorithms. A good understanding of the algorithms helps in better understanding and analyzing the results of the empirical studies.

In this chapter, we first briefly discuss the problem of prediction learning and the problem of control learning. We continue by discussing 10 off-policy prediction learning algorithms that are used to solve the prediction learning problem. We close the chapter by discussing the Q-leaning algorithm, which we will later use in the development of the QRC algorithm.

3.1 The Problem of Prediction Learning and the Problem of Control Learning

In solving the off-policy prediction learning problem, the agent's goal is to learn predictions in an off-policy manner, where predictions are formulated using value functions. Recall from Chapter 2 that, for all $s \in \mathcal{S}$,

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = s]\,, \tag{3.1}$$

where the expectation is subscripted by π to indicate that they are conditioned on π being followed. In off-policy learning, we assume that actions are taken according to a behavior policy b, while the agent seeks to learn the value

function under a different policy π. We assume both the target and the behavior policies are known and static, although of course in many applications of interest one or the other may be changing.

The prediction learning problem can be contrasted with the control learning problem which we will consider in Chapter 5. In off-policy control learning, the agent's goal is to find the optimal policy, rather than evaluating a fixed given policy. For example, in the most well-known off-policy control learning algorithm, Q-learning, the policy that the agent learns about is the greedy policy, while the policy that the agent follows is a more exploratory policy such as an ϵ-greedy, policy.

3.2 The Off-policy TD(λ) Algorithm

The value of a state $S_t = s$ can be written recursively, using the value of the next state S_{t+1}. To show this, we use the following identity:

$$\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X \mid Y]], \tag{3.2}$$

which in statistics, is known as the law of total expectation or the tower rule. In (3.1) we defined the value function, based on which we can write:

$$v_\pi(s') \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']. \tag{3.3}$$

Using (3.2) and (3.3), the value of the next state S_{t+1} can be written:

$$\begin{aligned}\mathbb{E}_\pi[v_\pi(S_{t+1}) \mid S_t = s] &= \mathbb{E}_\pi[\mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] \mid S_t = s] \\ &= \mathbb{E}_\pi[G_{t+1} \mid S_t = s],\end{aligned} \tag{3.4}$$

which is used below to show that the value of the current state $S_t = s$ is equal to the expectation of the reward plus the discounted value of the next state:

$$\begin{aligned}v_\pi(s) &\stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} \mid S_t = s] + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]. \qquad \text{(by (3.4))}\end{aligned}$$

The equation $v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$ is referred to as the *Bellman equation* for v_π. The Bellman equation says that the value of a state is equal to the expectation of the immediate reward plus the discounted value of the next state. If $v_\pi(s)$ is moved to the right hand side and to the inside of the expectation we get:

$$\mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) - v_\pi(s) \mid S_t = s] = 0, \tag{3.5}$$

which is referred to as the *Bellman error*. The error is equal to 0 for the true value function, v_π.

In its tabular form, temporal-difference learning updates the weight vector, $\mathbf{w}$, in the direction that minimizes the Bellman error. When using function approximation, (3.5) can often only hold approximately:

$$\mathbb{E}_\pi[R_{t+1} + \gamma \hat{v}_{\mathbf{w}}(S_{t+1}) - \hat{v}_{\mathbf{w}}(s) \mid S_t = s] \approx 0. \tag{3.6}$$

The simplest form of temporal-difference learning, the TD(0) algorithm, uses a sample of the left hand side of (3.6) to update $\mathbf{w}$. This sample is called the *TD-error* and is defined as:

$$\delta_t \stackrel{\text{def}}{=} R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t, \tag{3.7}$$

where linear function approximation is used to approximate the value function. TD(0) uses (3.7) to update the weight vector as follows:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{x}_t, \tag{3.8}$$

where α is a scalar constant step-size parameter. If α is adaptive and changes at each time step, it is denoted by α_t.

Off-policy TD(0) augments the TD(0) update, (3.8), with an importance sampling ratio:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \rho_t \alpha \delta_t \mathbf{x}_t. \tag{3.9}$$

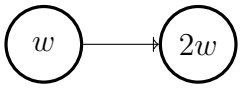
One of the widely known ideas to make TD-style algorithms more efficient, are *eligibility traces*. Eligibility traces help assign credit to features that

were activated in the past based on how far in the past they were activated. Eligibility traces assign credit by storing a fading trace of the features that were activated. In this dissertation, our focus is on one of the form of eligibility traces, called accumulating traces. The Off-policy TD(λ) algorithm with eligibility traces is fully specified by the following update rules:

$$\begin{aligned} \delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\ \mathbf{z}_t &\leftarrow \rho_t(\gamma\lambda \mathbf{z}_{t-1} + \mathbf{x}_t), \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\ \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t. \end{aligned} \tag{3.10}$$

Off-policy TD(λ) might diverge when combined with linear function approximation. A simple example can help see this intuitively (Sutton & Barto, 2018). Suppose, two states in an MDP, whose values are approximated using the same weight w, except that the second state has a value twice as big as the first state as shown below. In this example, w is a single number and the feature vectors are 1 for the left state and 2 for the right state. Suppose, for this MDP that $\gamma = 1$ and the rewards are 0 on all transitions.

If the transition from w to $2w$ is experienced repeatedly, the weight will diverge to infinity. Suppose, the first time the transition is experienced, $w = 10$. This means $2w$ is 20 and the TD error will be 10. If, for example, $\alpha = 0.5$, the value of w will be increased by 5 and will change to 15. If this transition is experienced repeatedly, it is clear that the weights will diverge to infinity.



There is no way on-policy TD(0) may diverge in the example above. In on-policy learning, when the agent moves from $2w$ to w, through some path not shown in the figure above, the weights will eventually decrease and convergence will be guaranteed. In off-policy learning, however, there can be situations where the weights do not decrease when taking the path from $2w$ to w and divergence will occur.

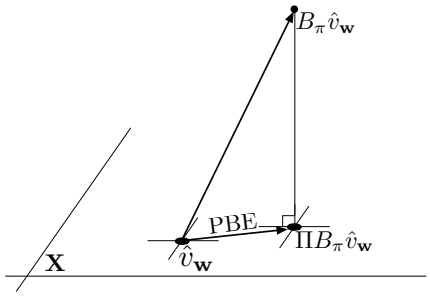


Figure 3.1: Geometry of linear function approximation shown in three dimensional space.

3.3 Gradient-TD Algorithms

One way to assure convergence is to use stochastic gradient descent to minimize an objective function. By using stochastic gradient descent, convergence will follow, even in the off-policy learning case. One important question is: What objective function should be minimized?

One option is to minimize the mean squared projected Bellman error, which we denote by $\overline{\text{PBE}}$. To define $\overline{\text{PBE}}$, we first need to define the Bellman operator. The Bellman error, in state s is:

$$\mathbb{E}_\pi[R_{t+1} + \gamma \hat{v}_{\mathbf{w}}(S_{t+1}) - \hat{v}_{\mathbf{w}}(S_t) \mid S_t = s]\,,$$

or more explicitly:

$$\left(\sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma \hat{v}_{\mathbf{w}}(s')] \right) - \hat{v}_{\mathbf{w}}(s), \tag{3.11}$$

where p is the underlying transition probability distribution of the MDP. The Bellman operator $B_\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$, at state s, is defined as:

$$(B_\pi \hat{v}_{\mathbf{w}})(s) \stackrel{\text{def}}{=} \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma \hat{v}_{\mathbf{w}}(s')]. \tag{3.12}$$

As seen in (3.12), the Bellman operator works irrespective of the parameters of the value function. This means after applying the Bellman operator to a value function, the new value function might not be representable in the feature space $\mathbf{X}$. It can, however, be projected back into the space of the representable functions, using a projection matrix. In the linear function approximation

case, the projection operator is linear, meaning that it can be represented as an $|\mathcal{S}| \times |\mathcal{S}|$ matrix:

$$\Pi \stackrel{\text{def}}{=} \mathbf{X}(\mathbf{X}^\top \mathbf{D}\mathbf{X})^{-1}\mathbf{X}^\top \mathbf{D},$$

where $\mathbf{X}$ is a matrix of size $|\mathcal{S}| \times d$ with feature vectors of all states as its rows and $\mathbf{D}$ being a diagonal matrix with the state visitation distribution on its diagonal. To minimize the distance between the value function and the projection of the value function after applying the Bellman operator, we first need to apply the Bellman operator to the current value function using (3.12). The Bellman error for all states, put together in a vector is called the Bellman error vector. We denote the Bellman error vector by $\bar{\delta}_{\mathbf{w}}$ ((3.11) for all s). This vector can be projected to the space of representable value function. We denote the projected vector by $\Pi\bar{\delta}_{\mathbf{w}}$. The distance between the value function before applying the Bellman operator and the projection of the function after applying the Bellman operator is known as the *Mean Squared Projected Bellman Error* or the $\overline{\text{PBE}}$ and can be minimized directly using stochastic gradient descent. See Figure 3.1. The $\overline{\text{PBE}}$ is defined in a condensed form as:

$$\overline{\text{PBE}}(\mathbf{w}) = ||\Pi\bar{\delta}_{\mathbf{w}}||^2_{\mu_b}, \tag{3.13}$$

where μ_b is the state visitation distribution induced by policy b.

All members of the Gradient-TD family minimize some form of the mean squared projected Bellman error. GTD(λ) and GTD2(λ) both minimize the $\overline{\text{PBE}}$ objective (Sutton et al. 2009). An earlier version of the Gradient-TD family minimizes the *Norm of the Expected TD Update*, or the NEU (Sutton, Maei, & Szepesvári, 2008). The NEU does not have a readily available geometric interpretation, but it is only different from the $\overline{\text{PBE}}$ in how the objective function (3.13) is weighted. The algorithm proposed by Sutton, Maei, & Szepesvári (2008) was called GTD. The algorithms proposed by Sutton et al. (2009) were called TDC and GTD2. Later on, in Maei (2011), the TDC and GTD2 algorithms were combined with eligibility traces and were called GTD(λ) and GTD2(λ), respectively. This means that GTD(0) is the same as TDC and GTD2(0) is the same as GTD2. Throughout the dissertation,

we use both TDC and GTD(0) names to refer to the same algorithm. The algorithm that minimizes the NEU was later found to be inferior to GTD and GTD2 algorithms and was abandoned (Sutton et al., 2009; Dann, Neumann, & Peters, 2014).

Update rules for GTD(λ) and GTD2(λ) are similar to Off-policy TD(λ). The only difference is that the Gradient-TD algorithms have a correction term that changes the original TD update to make sure it follows the direction of the gradient of $\overline{\text{PBE}}$ at each time step. To estimate the correction term, both algorithms use a secondary or auxiliary learned weight vector, which we denote by $\mathbf{v}$, that can be learned at a different rate than the primary weight vector. The update rules for GTD(λ) are:

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_t(\gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}_t) \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{v}_{t+1} &\leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}} \Big[\delta_t \mathbf{z}_t - (\mathbf{v}_t^\top \mathbf{x}_t) \mathbf{x}_t \Big] \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t - \underbrace{\alpha \gamma_{t+1}(1-\lambda_{t+1})(\mathbf{v}_t^\top \mathbf{z}_t)\mathbf{x}_{t+1}}_{\text{correction term}},
\end{aligned}$$

and the update rules for GTD2(λ) are:

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_t(\gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}_t) \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{v}_{t+1} &\leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}} \Big[\delta_t \mathbf{z}_t - (\mathbf{v}_t^\top \mathbf{x}_t) \mathbf{x}_t \Big] \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha (\mathbf{v}_t^\top \mathbf{x}_t) \mathbf{x}_t - \alpha \gamma_{t+1}(1-\lambda_{t+1})(\mathbf{v}_t^\top \mathbf{z}_t)\mathbf{x}_{t+1}.
\end{aligned}$$

There has been a few attempts at making GTD(λ) and GTD2(λ) faster. Maei (2011) showed that although Gradient-TD algorithms provide better convergence guarantees than off-policy TD(λ), they learn slower. HTD was the first attempt at making Gradient-TD algorithms faster. Hybrid TD was first proposed by Maei (2011). It was later developed further by Hackman (2012). It was finally extended to incorporate eligibility trace case by White and White (2016). HTD(λ) was an attempt at combining the fast learning of Off-policy TD(λ) and the convergence guarantees of the Gradient-TD family.

HTD(λ) is a generalization of TD(λ) in the sense that in the on-policy case, HTD(λ) reduces to TD(λ). To better understand HTD(λ), we first write the $\overline{\text{PBE}}$ in an alternative form and for the off-policy case and full bootstrapping:

$$\overline{\text{PBE}}(\mathbf{w}) = \mathbb{E}_b[\delta_t \rho_t \mathbf{x}_t]^\top \mathbb{E}_b\big[\mathbf{x}_t \mathbf{x}_t^\top\big]^{-1} \mathbb{E}_b[\delta_t \rho_t \mathbf{x}_t]\,,$$

We then use the following definitions:

$$\begin{aligned}
\mathbf{C} &\stackrel{\text{def}}{=} \mathbb{E}_b\big[\mathbf{x}_t \mathbf{x}_t^\top\big]\,,\\
\mathbf{A} &\stackrel{\text{def}}{=} -\mathbb{E}_b\big[(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t)\rho_t \mathbf{x}_t^\top\big]\,,\\
\mathbf{b} &\stackrel{\text{def}}{=} \mathbb{E}_b\big[R_{t+1}\rho_t \mathbf{x}_t^\top\big]\,,
\end{aligned}$$

where R_{t+1} is the reward and $\mathbf{x}$ is the feature vector of the state S_t. Using the above identities, we write the $\overline{\text{PBE}}$ in the following form (Hackman, 2012):

$$\overline{\text{PBE}} = (-\mathbf{A}\mathbf{w} + \mathbf{b})^\top \mathbf{C}^{-1}(-\mathbf{A}\mathbf{w} + \mathbf{b}), \tag{3.14}$$

where

$$\begin{aligned}
-\mathbf{A}\mathbf{w} + \mathbf{b} &= \mathbb{E}_b[\delta_t \rho_t \mathbf{x}_t]\,,\\
\mathbf{C} &= \mathbb{E}_b\big[\mathbf{x}_t \mathbf{x}_t^\top\big]\,.
\end{aligned}$$

In (3.14), the $\mathbf{C}$ matrix is weighting $-\mathbf{A}\mathbf{w} + \mathbf{b}$. As long as $-\mathbf{A}\mathbf{w} + \mathbf{b}$ becomes $\mathbf{0}$ asymptotically, the algorithms find a solution to the $\overline{\text{PBE}}$ and the weighting, $\mathbf{C}$ is irrelevant in the quality of the solution. However, the rate of convergence might change with $\mathbf{C}$. More specifically, the $\mathbf{C}$ matrix can be replaced by any positive definite matrix and the resulting algorithm is guaranteed to be stable and will converge to the minimum of the $\overline{\text{PBE}}$. The HTD(λ) algorithm replaces $\mathbf{C}^{-1}$ with $\mathbf{A}_b^{-\top}$, where:

$$\mathbf{A}_b^{-\top} \stackrel{\text{def}}{=} \mathbb{E}_b\big[\big(\mathbf{x}_t - \gamma \mathbf{x}_{t+1}\big)\mathbf{x}_t^\top\big]\,,$$

Using $\mathbf{A}_b^{-\top}$ instead of $\mathbf{C}^{-1}$, and computing the derivative of the resulting $\overline{\text{PBE}}$, gives the HTD(0) update rules.

White and White, (2016) extended HTD to the case of general bootstrapping. HTD(λ) is fully specified by the following update rules:

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_t(\gamma\lambda \mathbf{z}_{t-1} + \mathbf{x}_t) \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{z}_t^{\mathbf{b}} &\leftarrow \gamma\lambda \mathbf{z}_{t-1}^{\mathbf{b}} + \mathbf{x}_t \qquad \text{with } \mathbf{z}_{-1}^{\mathbf{b}} = \mathbf{0} \\
\mathbf{v}_{t+1} &\leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}} \Big[\delta_t \mathbf{z}_t - (\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})(\mathbf{v}_t^\top \mathbf{z}_t^{\mathbf{b}}) \Big] \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \Big[\delta_t \mathbf{z} + (\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})(\mathbf{z}_t - \mathbf{z}_t^{\mathbf{b}})^\top \mathbf{v}_t \Big],
\end{aligned}$$

where $\mathbf{z}_t^{\mathbf{b}}$ is an eligibility trace vector that does not include importance sampling ratios and $\mathbf{z}_t$ is the normal off-policy trace with the importance sampling ratio.

Proximal Gradient-TD algorithms are another set of algorithms proposed to improve on the original Gradient-TD algorithms (Mahadevan et al., 2014; Liu et al., 2015; Liu et al., 2016). Proximal Gradient-TD algorithms improve on classic Gradient-TD algorithms in the sense that they use the true gradient of the $\overline{\text{PBE}}$ objective to update the weight vector. This is in contrast to Gradient-TD algorithms that are not true stochastic gradient algorithms with respect to the $\overline{\text{PBE}}$ objective. The reason why the classic Gradient-TD algorithms are not exactly stochastic gradient descent is that they include a product of expectations over the next feature vector in the gradient of the objective function. In the on-policy case:

$$\begin{aligned}
-\frac{1}{2}\nabla\overline{\text{PBE}}(\mathbf{w}) &= \mathbb{E}\left[(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})\mathbf{x}^\top\right] \mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^\top\right]^{-1} \mathbb{E}\left[\delta_t \mathbf{x}_t\right] && (3.15) \\
&= \mathbf{A}^\top \mathbf{C}^{-1} \left(-\mathbf{A}\mathbf{w} + \mathbf{b}\right). && (3.16)
\end{aligned}$$

The next state's feature vector, $\mathbf{x}_{t+1}$, appears in $\mathbf{A}$. The gradient of $\overline{\text{PBE}}$ in (3.16) multiplies $\mathbf{A}$ with itself and includes a product of expectations of the next state's feature vector. To get an unbiased sample of the product, two independent samples are required. However, during the normal interaction of the agent with the environment, it is only possible to get one sample. Gradient-TD algorithms get around this issue by learning a second weight vector, $\mathbf{v}$, and forming a quasi-stationary estimate of the last two expectations in (3.15).

Proximal Gradient-TD algorithms approach the double sampling challenge by writing the objective function using a saddle-point formulation:

$$\overline{\text{PBE}}(\mathbf{w}) = \min_{\mathbf{w}} \max_{\mathbf{v}} (\mathbf{b} - \mathbf{A}\mathbf{w})^\top \mathbf{v} - \tfrac{1}{2}\|\mathbf{v}\|^2_{\mathbf{C}}, \tag{3.17}$$

where $\frac{1}{2}\|\mathbf{v}\|^2_{\mathbf{C}}$ is the norm of $\mathbf{v}$ weighted by $\mathbf{C}$, or simply $\mathbf{v}^\top\mathbf{C}\mathbf{v}$. Computing the derivative of the objective function (3.17), with respect to $\mathbf{v}$, results in:

$$\nabla_{\mathbf{v}}\overline{\text{PBE}} = \mathbf{b} - \mathbf{A}\mathbf{w} - \mathbf{C}\mathbf{v},$$

and the derivative with respect to $\mathbf{w}$ results in:

$$\nabla_{\mathbf{w}}\overline{\text{PBE}} = -\mathbf{A}^\top\mathbf{v}.$$

The gradient of the saddle-point formulated objective does not have the product of the $\mathbf{A}$ matrix with itself and avoids the double sampling issue. This means the algorithm that minimizes the saddle-point objective is a true stochastic gradient descent algorithm and as a result, they can make use of algorithms developed for improving the convergence rate of stochastic gradient descent. Mahadevan et al. (2014) used stochastic mirror-prox (Juditsky, Nemirovski, & Tauvel, 2011) to derive a new version of Gradient-TD family, called Proximal GTD2. Proximal GTD2(λ) is fully specified by the following equations:

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top\mathbf{x}_{t+1} - \mathbf{w}_t^\top\mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_t(\gamma_t\lambda_t\mathbf{z}_{t-1} + \mathbf{x}_t) \quad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{v}_{t+\frac{1}{2}} &\leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}}\Big[\delta_t\mathbf{z}_t - (\mathbf{v}_t^\top\mathbf{x}_t)\mathbf{x}_t\Big] \\
\mathbf{w}_{t+\frac{1}{2}} &\leftarrow \mathbf{w}_t + \alpha(\mathbf{v}_t^\top\mathbf{x}_t)\mathbf{x}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\mathbf{v}_t^\top\mathbf{z}_t)\mathbf{x}_{t+1} \\
\delta_{t+\frac{1}{2}} &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_{t+\frac{1}{2}}^\top\mathbf{x}_{t+1} - \mathbf{w}_{t+\frac{1}{2}}^\top\mathbf{x}_t \\
\mathbf{v}_{t+1} &\leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}}\Big[\delta_{t+\frac{1}{2}}\mathbf{z}_t - (\mathbf{v}_{t+\frac{1}{2}}^\top\mathbf{x}_t)\mathbf{x}_t\Big] \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha(\mathbf{v}_{t+\frac{1}{2}}^\top\mathbf{x}_t)\mathbf{x}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\mathbf{v}_{t+\frac{1}{2}}^\top\mathbf{z}_t)\mathbf{x}_{t+1}.
\end{aligned}$$

There are other objective functions that can be minimized. One alternative objective function is the Mean Squared Bellman Error ($\overline{\text{BE}}$). $\overline{\text{BE}}$ is similar to the $\overline{\text{PBE}}$ objective but does not have the projection operator. The so-called

residual gradient algorithms minimize $\overline{\text{BE}}$. There has been some recent work in minimizing the $\overline{\text{BE}}$ objective using stochastic gradient descent (Dai et al., 2018; Feng, Li, & Liu, 2019). These algorithms are true stochastic gradient descent algorithms and in turn provide strong convergence guarantees; however, we choose not to consider such algorithms due to reasons such as non-learnability. See Sutton and Barto (2018) for a thorough discussion of why minimizing the $\overline{\text{BE}}$ might not be desirable.

Before closing the Gradient-TD section, we would like to mention that the $\overline{\text{PBE}}$, which is used as an objective function, can naturally be used to measure the error at each time step. To do so, we use (3.14). To compute the $\mathbf{A}$ matrix, $\mathbf{C}$ matrix, and the $\mathbf{b}$ vector, we use the following identities from White (2015):

$$\mathbf{C} \stackrel{\text{def}}{=} \mathbf{X}^\top \mathbf{D} \mathbf{X},$$
$$\mathbf{A} \stackrel{\text{def}}{=} \mathbf{X}^\top \mathbf{D} (\mathbf{I} - \gamma \mathbf{P}^\pi) \mathbf{X},$$
$$\mathbf{b} \stackrel{\text{def}}{=} \mathbf{X}^\top \mathbf{D}\, \mathbf{r}^\pi.$$

The vector $\mathbf{r}^\pi \in \mathbb{R}^{|\mathcal{S}|}$ is a column vector with each component equal to the expectation of one step reward under the target policy, $\mathbb{E}_\pi[R_{t+1} \mid S_t = s]$. As the reader might remember, the matrix $\mathbf{D} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$ is a diagonal matrix whose diagonal elements correspond to the stationary distribution under the behavior policy: $\mathbf{D}(s, s) \stackrel{\text{def}}{=} d_b(s)\ \forall s \in \mathcal{S}$, which can be approximated from data, $\mathbf{X} \in \mathbb{R}^{|\mathcal{S}| \times d}$ is a matrix with $|\mathcal{S}|$ rows and d columns: one row for each $s \in |\mathcal{S}|$ where each state is represented with a d-dimensional feature vector. Finally, $P^\pi : \mathcal{S} \times \mathcal{S} \to [0,\, 1]$ where $P^\pi(s, s') \stackrel{\text{def}}{=} \sum_{a \in \mathcal{A}} \pi(s, a) p(s, a, s')$, which can be approximated given access to the parameters of the MDP. The function $p(s, a, s')$ is the one-step transition dynamics of the MDP: probability of transitioning into s' when action a is taken in state s.

3.4 Emphatic-TD Algorithms

Emphatic-TD algorithms provide an alternative strategy for stable off-policy learning (Sutton, Mahmood, & White, 2016). Gradient-TD algorithms correct the semi-gradient updates of TD(λ) so that the updates are in the direction

of the gradient and convergence guarantees follow. Emphatic-TD algorithms, on the other hand, use semi-gradient updates, just like Off-policy TD(λ).

The main idea of Emphatic-TD algorithms is to emphasize and de-emphasize the update at different time steps. By using emphasis, Emphatic-TD algorithms assure that selective updating in off-policy learning cannot cause divergence. This can be explained by going over the $w-2w$ example. As we noted, the problem is that the parameter is updated when the agent moves from w to $2w$ but no updates happen when the agent moves from $2w$ to w. If an algorithm assures updates in parameter happen when moving from $2w$ to w, the value of w will decrease and divergence can be prevented. This is exactly what Emphatic-TD algorithms do. Emphatic-TD algorithms assure that the value of a state (or parameter vector) will be updated if the state is reachable from another state for which we update the parameter vector. In the $w-2w$ example, with Emphatic-TD algorithms, if an update happens when the agent moves from w to $2w$, an update is assured when the agent moves from $2w$ to w.

The first Emphatic-TD algorithm was Emphatic TD(λ) (Sutton, Mahmood, & White, 2016). This algorithm is sometimes referred to as ETD(λ). It is the first algorithm with convergence guarantees under off-policy training that has one set of learned weight vector and one step-size parameter. The Emphatic TD(λ) algorithm is fully specified described by the following equations:

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
F_t &\leftarrow \rho_{t-1}\gamma_t F_{t-1} + 1 \qquad \text{with } F_{-1} = 0 && (3.18)\\
M_t &\leftarrow \lambda + (1-\lambda)F_t && (3.19)\\
\mathbf{z}_t &\leftarrow \rho_t \left(\gamma_t \lambda \mathbf{z}_{t-1} + M_t \mathbf{x}_t\right) \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} && (3.20)\\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t,
\end{aligned}$$

where F_t in (3.18) is the *followon trace* and M_t is called the *emphasis*. As λ approaches 1, M_t approaches $\lambda = 1$. At the extreme, when $\lambda = 1$, we have $M_t = \lambda = 1$ in (3.19), M_t disappears from (3.20), and Emphatic TD(1) will

reduce to Off-policy TD(1). As a result, as $\lambda \to 1$, we might expect Emphatic TD(λ) to behave more like Off-policy TD(λ). This also means that the largest difference between TD(λ) and Emphatic TD(λ) should probably be expected at $\lambda = 0$.

Emphatic TD(λ) is prone to high variance. In spite of correcting for the difference between the target and behavior policies at each time step, Emphatic TD(λ) also corrects for the differences between the policies in the past. To do so, it uses a product of importance sampling ratios, as shown in (3.18). As a result, the followon trace can become large over time (even unbounded) and the step-size parameter should be reduced further down to avoid divergence.

Later on, Emphatic TD(λ, β) was proposed to reduce the variance of Emphatic TD(λ). As the name suggests, the algorithm has an extra parameter β. This parameter provides some control over how quickly the magnitude of F_t grows. All the update rules for Emphatic TD(λ, β) are the same as the ones for Emphatic TD(λ), except for the update to the followon trace:

$$F_t \leftarrow \ \rho_{t-1}\beta F_{t-1} + 1. \tag{3.21}$$

By comparing (3.18) and (3.21) we see that if β is set to a value smaller than γ, F_t will grow at a slower rate than when $\beta = \gamma$. If $\beta = 0$ in (3.21), Emphatic TD(λ, β) reduces to Off-policy TD(λ), and if β is set to γ, the algorithm reduces to Emphatic TD(λ).

The emphasis idea has use-cases other than assuring convergence under off-policy training. Although originally proposed for off-policy learning, Emphatic TD(λ) does not reduce to TD(λ) even if the target and behavior policies are the same. In fact, Emphatic TD(λ) is shown to empirically outperform TD(λ) in some on-policy experiments (Ghiassian, Rafiee, & Sutton, 2016). It has also been shown that Emphatic TD(λ) can solve some of the counterexamples to the TD(λ) algorithms in the on-policy case (Gu, Ghiassian, & Sutton, 2019). It is also notable that the emphasis idea can be combined with other families of algorithms, such as the Gradient-TD family.

3.5 Algorithms for Fast Off-policy Prediction Learning

A common concern with off-policy learning is that large importance sampling ratios might cause high variance. Several methods have been proposed that avoid large importance sampling ratios. Tree Backup(λ) (Precup, Sutton, & Singh, 2000), Retrace(λ) (Munos et al., 2016), and ABQ(ζ) (Mahmood, Yu, & Sutton, 2017) all avoid large importance sampling ratios. Tree Backup(λ), and ABQ(ζ) avoid explicit use of importance sampling ratios in their update rules. Retrace(λ), simply truncates any importance sampling ratio that happens to be larger than one.

Tree Backup(λ), Retrace(λ), and ABQ(ζ) can all be seen as Off-policy TD(λ) with λ generalized from a constant to a function of state and action. This unification was highlighted by Mahmood, Yu, and Sutton (2017) and also by Yu, Mahmood, and Sutton (2018). The unification makes it simple to explain all three algorithms: all methods are the same as Off-policy TD(λ) but each method uses a different action-dependent trace function $\lambda : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Simply put, each method sets λ differently at each time step. All three methods mentioned above were originally proposed for control. Later in this dissertation, we provide an easy way to understand all three algorithms and also present the natural state-value variants of all these algorithms.

Between the three algorithms, Retrace(λ) was later extended to prediction setting by Espeholt et al. (2018). The resulting algorithm is called V-trace(λ). Here, we use a simplified version of the V-trace algorithm that simply truncates the importance sampling ratio in the eligibility trace. The V-trace algorithm used here uses the following update rules:

$$\begin{aligned}
\delta_t^\rho &\stackrel{\text{def}}{=} \rho_t \Big(R_{t+1} + \gamma_{t+1} \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \Big) \\
\mathbf{z}_t &\leftarrow \max(\rho_{t-1}, 1) \left(\gamma_t \lambda_t \mathbf{z}_{t-1}\right) + \mathbf{x}_t \quad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t.
\end{aligned}$$

3.6 Least-squares Algorithms

Temporal-difference learning algorithms, when they converge, satisfy the Bellman equation (Sutton & Barto, 2018); a fixed point solution that can be directly computed with least-squares algorithms (Bradtke & Barto, 1996; Boyan, 1999). In fact, TD(λ) and Gradient-TD algorithms converge to the minimum of the mean squared projected Bellman error ($\overline{\text{PBE}}$), also known as the TD(λ) fixed point. In the case of a fixed basis, we can analytically solve the $\overline{\text{PBE}}$ for the weights that satisfy the fixed point equation. The algorithm that solves the least-squares problem to find the fixed point of TD(λ) is called least Squares Temporal-Difference or LSTD(λ). The weight vector computed by LSTD(λ) (from a finite batch of training data) represents the weight vector that TD(λ) would converge to with repeated presentation of online data under standard conditions.

Different weightings of the $\overline{\text{PBE}}$ can be simply incorporated into LSTD(λ). For example, combining LSTD(λ) and the emphatic weighting produces the weight vector that Emphatic TD(λ) would converge to, given a fixed batch of data. Although, LSTD(λ) can be updated online and incrementally, its complexity is different from the algorithms that we used in our empirical study in this work. Least-squares algorithms are only of interest as baselines in this work because their computation is quadratic in the number of weights ($O(n^2)$) and are in general problematic with large state representations (like for example representations that might be learned using a neural network), compared to the algorithms that are presented so far in the work such as Gradient-TD methods that require linear computation per time step in the number of weights ($O(n)$).

To compute the TD(λ) fixed point, LSTD(λ) approximates $\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$ from the data. The values of the $\mathbf{A}$ matrix and the $\mathbf{b}$ vector can be estimated

incrementally using the following update rules:

$$\begin{aligned}
\mathbf{z}_t &\leftarrow \rho_t \gamma_t (\lambda_t \mathbf{z}_{t-1} + \mathbf{x}_t) \\
\mathbf{A}_{t+1} &\leftarrow \mathbf{A}_t + \frac{1}{t+1}[\mathbf{z}_t(\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})^\top - \mathbf{A}_t] \\
\mathbf{b}_{t+1} &\leftarrow \mathbf{b}_t + \frac{1}{t+1}[R_{t+1}\mathbf{z}_t - \mathbf{b}_t].
\end{aligned}$$

where t is the time starting from 0, $\mathbf{z}$ is the eligibility trace and ρ is the importance sampling ratio.

We can similarly find the LSTD(λ) fixed-point with emphatic weighting by simply changing the trace update rule mentioned above, to:

$$\begin{aligned}
F_t &\leftarrow \beta \rho_{t-1} F_{t-1} + I_t \\
M_t &\leftarrow \lambda_t I_t + (1 - \lambda_t) F_t \\
\mathbf{z}_t &\leftarrow \rho_t (\gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}_t M_t).
\end{aligned}$$

where F_t is the followon trace at time step t, and M_t is the emphasis.

3.7 Q-learning

Q-learning is perhaps one of the oldest off-policy learning algorithms (Watkins, 1989). The Q-learning algorithm is off-policy because the policy that is being learned (the target policy) is the optimal policy, while the behavior policy is typically an exploratory policy (e.g., ϵ-greedy) with respect to the current value function. This means that the value function that is being learned is different from the policy that is used for behavior. The update rules for the Q-learning algorithm are:

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma \hat{q}(S_{t+1}, a') - \hat{q}(S_t, A_t) \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{x}(S_t, A_t),
\end{aligned} \tag{3.22}$$

where a' is the action that the policy we are evaluating would take in state S_{t+1}. Q-learning learns the greedy policy, and so $a' = \arg\max_a q(S_{t+1}, a)$ and $\delta_t = R_{t+1} + \gamma \max_a q(S_{t+1}, a) - q(S_t, A_t)$. This action a' may differ from the (exploratory) action A_{t+1} that is actually executed, and so this estimation is

off-policy. As discussed previously in Chapter 2, the action-value function, $\hat{q}$ is defined as the dot product of the weight vector and the state-action feature vector.

3.8 Summary

In this chapter, we explained the off-policy prediction learning algorithms that we will use in the empirical studies of this dissertation. We started with a discussion of Off-policy TD(λ) algorithm and how it might diverge. This motivated the introduction of Gradient-TD and Emphatic-TD family of algorithms that are guaranteed to converge under off-policy training. We discussed the objective function that Gradient-TD algorithms minimize, the $\overline{\text{PBE}}$, and various approaches to minimizing this objective function. We then discussed algorithms that are proposed for fast off-policy learning. Finally, we closed the discussion of algorithms by providing a brief summary of how Least-squares algorithms can use the data to find an approximate solution to the minimum of the $\overline{\text{PBE}}$, or an approximate solution to the minimum of the emphatic weighted $\overline{\text{PBE}}$.

Chapter 4

Temporal-Difference Learning with Regularized Corrections[1]

This dissertation makes five main contributions. It introduces three new algorithms, and conducts two empirical studies. This chapter presents the first new algorithm: TDRC. All three algorithms proposed in this thesis have a common goal: improve the practicality of online off-policy prediction. The TDRC algorithm is not an exception. The goal of TDRC is to take a step in closing the performance gap that has been shown to exist between Off-policy TD(λ) and Gradient-TD algorithms.

The TDRC algorithm is similar to the TDC (also known as GTD(0)) algorithm, with the difference that the second weight vector is regularized in TDRC but not in TDC. We show that with more regularization, TDRC acts like TD, and with no regularization, it reduces to TDC. We find that for an interim level of regularization, TDRC obtains the best of both algorithms, and is quite insensitive to the regularization parameter: a regularization parameter of 1.0 was effective across all experiments. We show that our method (1) outperforms other Gradient-TD algorithms overall across a variety of problems, and (2) matches TD when TD performs well while maintaining convergence guarantees.

[1]The contents of this chapter are based on a paper co-authored by this author (Ghiassian et al., 2020).

4.1 Motivation

Both Off-policy TD and Q-learning, have well documented convergence issues, as highlighted in the seminal counterexample by Baird (1995). The fundamental issue is the combination of function approximation, off-policy updates, and bootstrapping: an algorithmic strategy common to sample-based TD learning and Dynamic Programming algorithms (Precup, Sutton, & Dasgupta, 2001). This combination can cause the value estimates to grow without bound (Sutton & Barto, 2018). Baird's result motivated over a decade of research and several new off-policy algorithms. The most well-known of these approaches, the Gradient-TD algorithms (Sutton et al., 2009), make use of a second set of weights and importance sampling.

Although sound under function approximation, these Gradient-TD algorithms are not commonly used in practice, likely due to the additional complexity of tuning two learning rate parameters. Many practitioners continue to use unsound approaches such as TD and Q-learning for good reasons. The evidence of divergence is based on highly contrived toy counter-examples. Often, many large scale off-policy learning systems are designed to ensure that the target and behavior policies are similar, and therefore less off-policy. However, if agents could learn from a larger variety of data streams, our systems could be more flexible and potentially more data efficient. Unfortunately, it appears that current architectures are not as robust under these more aggressive off-policy settings (van Hasselt et al., 2018). This results in a dilemma: the easy-to-use and typically effective TD algorithm can sometimes fail, but the sound Gradient-TD algorithms can be difficult to use.

4.2 The TDRC Algorithm

We develop a new algorithm, called TD with Regularized Corrections (TDRC). The idea is simple: regularize the update to the secondary parameters $\mathbf{v}$. The inspiration for the algorithm comes from the behavior of TDC observed by Maei (2011) and the behavior observed in experiments that we will discuss in

Section 4.3. Consistently, we find that TDC outperforms—or is comparable to—GTD2 in terms of optimizing the $\overline{\text{PBE}}$. These results match previous experiments comparing these two algorithms (White & White, 2016; Ghiassian et al., 2018). Previous results suggested that TDC could match TD (White & White, 2016); however, as we highlight in Section 4.3, this is only when the second step-size parameter is set so small that TDC is effectively behaving like TD. This behavior is unsatisfactory because to have guaranteed convergence, for example on Baird's Counterexample, the second step-size parameter needs to be large. Further, it is somewhat surprising that attempting to obtain an estimate of the gradient of the $\overline{\text{PBE}}$, as done by TDC, can perform so much more poorly than TD.

Notice that the $\mathbf{v}$ update is simply a linear regression update for estimating the (changing) target δ_t for both GTD2 and TDC (Sutton & Barto, 2018, Chapter 11). As $\mathbf{w}$ converges, δ_t approaches zero, and consequently $\mathbf{v}$ goes to $\mathbf{0}$ as well. But, a linear regression estimate of $\mathbb{E}[\delta_t|S_t = s]$ is not necessarily the best choice. In fact, using ℓ_2 regularization (ridge regression) can provide a better bias-variance trade-off: it can reduce variance without incurring too much bias. This is in particular true for $\mathbf{v}$, where asymptotically $\mathbf{v} = \mathbf{0}$ and so the bias disappears.

This highlights a potential reason that TD frequently outperforms TDC and GTD2 in experiments: the variance of $\mathbf{v}$. If TD already performs well, it is better to simply use the zero variance but biased estimate $\mathbf{v}_t = \mathbf{0}$. Adding ℓ_2 regularization with parameter β, i.e. $\beta\|\mathbf{v}\|_2^2$, provides a way to move between TD and TDC. For a very large β, $\mathbf{v}$ will be pushed close to zero and the update to $\mathbf{w}$ will be lower variance and more similar to the TD update. On the other hand, for $\beta = 0$, the update reduces to TDC and the estimator $\mathbf{v}$ will be an unbiased estimator with higher variance.

The resulting update equations for TDRC are

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha\rho_t\delta_t\mathbf{x}_t - \alpha\rho_t\gamma(\mathbf{v}_t^\top\mathbf{x})\mathbf{x}_{t+1}$$
$$\mathbf{v}_{t+1} \leftarrow \mathbf{v}_t + \alpha\big[\rho_t\delta_t - (\mathbf{v}_t^\top\mathbf{x}_t)\big]\mathbf{x}_t - \alpha\beta\mathbf{v}_t.$$

The update to $\mathbf{w}$ is the same as TDC, but the update to $\mathbf{v}$ now has the

Box 1: Deriving the update rule for the second weight vector of TDC and TDRC

TDRC differs from TDC in its update of the secondary weight vector. TDC's second weight vector seeks to find the weight vector $\mathbf{v}$ such that the following objective function is minimized:

$$J(\mathbf{v}) = \left(\mathbf{v}^\top \mathbf{x}_t - \delta_t\right)^2, \tag{4.1}$$

taking the derivative with respect to $\mathbf{v}$:

$$\begin{aligned}\nabla_{\mathbf{v}} &= 2(\mathbf{v}^\top \mathbf{x}_t - \delta_t)\nabla_{\mathbf{v}}(\mathbf{v}^\top \mathbf{x}_t - \delta_t), \\ &= 2(\mathbf{v}^\top \mathbf{x}_t - \delta_t)\mathbf{x}_t,\end{aligned}$$

which results in the following update for the secondary weight vector for TDC:

$$\mathbf{v}_{t+1} \leftarrow \mathbf{v}_t + \alpha\Big[\delta_t \mathbf{x}_t - (\mathbf{v}_t^\top \mathbf{x}_t)\mathbf{x}_t\Big].$$

TDRC uses the same rationale, with the difference that it uses ℓ_2 regularization in its objective:

$$J(\mathbf{v}) = \left(\mathbf{v}^\top \mathbf{x}_t - \delta_t\right)^2 + \beta||\mathbf{v}||_2^2,$$

the derivative of which is:

$$\nabla_{\mathbf{v}} = 2(\mathbf{v}^\top \mathbf{x}_t - \delta_t)\mathbf{x}_t + 2\beta||\mathbf{v}||.$$

Setting $\beta = 1$, results in the following update for TDRC's secondary weight vector:

$$\mathbf{v}_{t+1} \leftarrow \mathbf{v}_t + \alpha\Big[\delta_t \mathbf{x}_t - (\mathbf{v}_t^\top \mathbf{x}_t)\mathbf{x}_t\Big] - \alpha\mathbf{v}_t. \tag{4.2}$$

additional term $\alpha\beta\mathbf{v}_t$ which corresponds to the gradient of the ℓ_2 regularizer. Box 1 shows how we arrived at the update for the secondary weight vector $\mathbf{v}$. The updates only have a single shared step-size parameter, α, rather than a separate step-size parameter for the secondary weights $\mathbf{v}$. We make this choice precisely for our motivated reason upfront: for ease of use. Further, we find empirically that this choice is effective, and that the reasons for TDC's sensitivity to the second step-size parameter are mainly due to the fact that a small second step size enables TDC to behave like TD. Because TDRC has

this behavior by design, a shared step-size parameter is more effective.

Note that the ℓ_2 regularizer biases the estimator $\mathbf{v}$ towards $\mathbf{v} = \mathbf{0}$, the known optimum of the learning system as $\mathbf{w}$ converges. This means that the bias imposed on $\mathbf{v}$ disappears asymptotically, changing only the transient trajectory (we prove this in Theorem 4.4.1).

As a final remark, we motivate that TDRC should not require a second step-size parameter, but have introduced a new parameter (β) to obtain this property. The idea, however, is that TDRC should be relatively insensitive to β. The choice of β sweeps between two reasonable algorithms: TD and TDC. If we are already comfortable using TD, then it should be acceptable to use TDRC with a larger β. A smaller β will still result in a sound algorithm, though its performance may suffer due to the variance of the updates in $\mathbf{v}$. In our experiments, we in fact find that TDRC performs well for a wide range of β, and that our default choice of $\beta = 1$ works reasonably across all the problems that we tested.

4.3 Experiments in the Prediction Setting

We first establish the performance of TDRC across several small linear prediction tasks where we carefully sweep over parameters, analyze sensitivity, and average over many runs. The goal is to understand if TDRC has similar performance to TD, with similar parameter sensitivity, but avoids divergence. Before running TDRC, we set $\beta = 1$ across all the experiments to refrain from tuning this additional parameter. Code for all experiments is available at: https://github.com/rlai-lab/Regularized-GradientTD.

In the prediction setting, we investigate three different problems with variations in feature representations, target, and behavior policies. We choose problems that have been used in prior work to empirically investigate TD methods.

The first problem, Boyan's chain (Boyan, 2002), is a 14 state Markov chain where each state is represented by a compact feature representation. This encoding causes inappropriate generalization during learning, but v_π can be

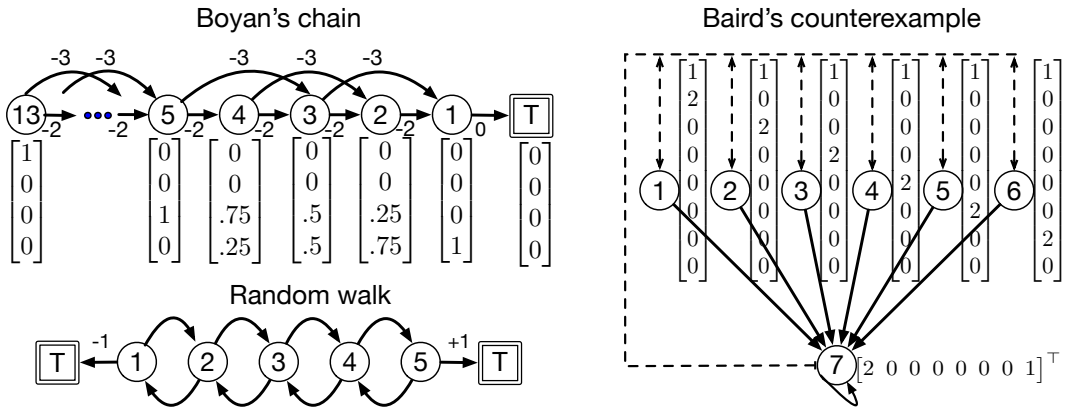


Figure 4.1: A graphic depiction of each of the three MDPs and the corresponding feature representations used in our experiments. We omit the three feature representations used in the Random Walk due to space restrictions (see Sutton et al., 2009). All unlabeled transitions emit a reward of zero.

represented perfectly with the given features. The task is undiscounted and the rewards are either -2 or -3 at each time step depending on the selected action (the reward on the last time step from state 1 to the terminal state is 0). The Boyan's chain is shown at the upper left side of Figure 4.1.

In Boyan's chain, each episode started at the leftmost state and the policy was to choose one of the two actions with equal probability in all states except state 1. In state 1, the one available actions was chosen with probability 1.

The second problem that we considered is Baird's (1995) well-known star counterexample. In this MDP, the target and behavior policy are very different resulting in large importance sampling corrections. Baird's counterexample has been used extensively to demonstrate the soundness of Gradient-TD algorithms, so provides a useful testbed to demonstrate that TDRC does not sacrifice soundness for ease of use. In Baird's counterexample, the discount factor is $\gamma = 0.99$ and all the rewards are 0. Baird's counterexample is shown on the right side of Figure 4.1.

In Baird's counterexample, the target policy is $\pi(solid|\cdot) = 1$, while the behavior policy is: $b(dashed|\cdot) = 6/7$ and $b(solid|\cdot) = 1/7$, where solid and dashed refer to the lines in Figure 4.1.

Finally, we include a seven state Random Walk MDP, with five non-

terminal states and two terminal states, one at each end (see the bottom left panel of Figure 4.1). This problem was originally studied by Sutton et al. (2009) to compare GTD2, TDC, and Off-policy TD. Similar to the original problem, we used three different feature representations: Tabular (unit basis vectors), Inverted, and Dependent features. Inverted features were proposed such that they cause extensive inappropriate generalization between states. Using Inverted features, the second state is for example represented by $\mathbf{x}_2 = (\frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})^\top$. The value $\frac{1}{2}$ was chosen such that the feature vector has unit norm. Dependent features had 3 dimensions, and the states were represented as follows: $\mathbf{x}_1 = (1, 0, 0)^\top$, $\mathbf{x}_2 = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)^\top$, $\mathbf{x}_3 = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})^\top$, $\mathbf{x}_4 = (0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})^\top$, and $\mathbf{x}_5 = (0, 0, 1)^\top$. This task is undiscounted and the rewards are all 0, except for transitioning to the terminal state for which the agent receives a -1 or +1 rewards, depending on the terminal state.

The only difference between the task studied here and the one studied by Sutton et al. (2009) is the behavior policy. The experiments conducted by Sutton et al. (2009) were on policy, whereas, like Hackman (2012), we used an off-policy variant of the problem. The agent starts each episode in the middle state. The behavior policy chooses the left and right action with equal probability, and the target policy chooses the right action 60% of the time.

We applied GTD2, TDRC, TD, HTD, TDRC, and V-trace to the problems discussed above for 3000 time steps and 200 independent runs. We swept over free parameters for every method comparing the parameters which performed best according to the area under the $\sqrt{\overline{\text{PBE}}}$ learning curve. The step-size parameters swept for all algorithms were $\alpha \in \{2^{-7}, 2^{-6}, \ldots, 2^0\}$. For TDC and HTD, we swept values of the second step-size parameter by sweeping over a multiplicative constant times the primary step-size parameter, $\eta \in \{2^0, 2^1, \ldots, 2^6\}$ maintaining the convergence guarantees of the two-timescale proof of convergence for TDC. For GTD2, we swept values of $\eta \in \{2^{-6}, 2^{-5}, \ldots, 2^5, 2^6\}$ as the saddle-point formulation of GTD2 allows for a much broader range of η while still maintaining convergence.

In the first set of results discussed below, we used the AdaGrad (Duchi, Hazan & Singer, 2011) algorithm to adapt a vector of step-size parameters

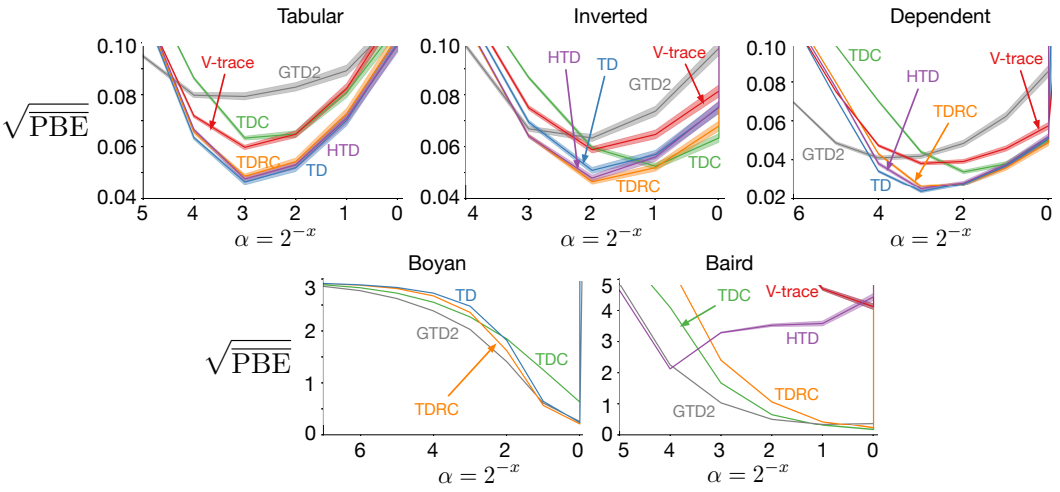


Figure 4.2: Step-size parameter sensitivity measured using average area under the the $\sqrt{\overline{\text{PBE}}}$ learning curve for each method on each problem. HTD and V-Trace are not shown in Boyan's Chain because they reduce to TD for on-policy problems. All algorithms that had more than one step-size parameter were free to choose the second step-size parameter that minimized the area under the learning curve for each first step-size parameter.

for each algorithm (Figures 4.2, 4.3, 4.4, 4.5). Additional results for constant scalar step-size parameters are provided later. Combining AdaGrad and Gradient-TD algorithms is straightforward. In gradient-based learning, AdaGrad uses statistics from the gradient vectors observed so far, to determine the size and direction of the update in each direction at each time step. When AdaGrad is applied to Gradient-TD learning, or any other temporal-difference learning algorithm, it determines the size and direction of the update at each time step using statistics from the direction of the update suggested by the original temporal-difference algorithm, such as TDC.

We provide the parameter sensitivity plots in Figure 4.2. Additionally, we report the performance with the best step-size parameter in Figure 4.3. In the bar plot, we compactly summarized relative performance to TDRC. TDRC performed well across problems, while every other method had at least one setting where it was noticeably worse than TDRC. TDC generally performed much better than GTD2. This is most probably because TDC learned faster than GTD2, as both algorithms converge to the same asymptotic error level.

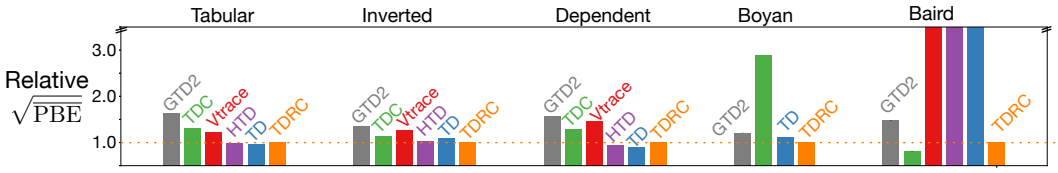


Figure 4.3: The normalized average area under the $\sqrt{\overline{\text{PBE}}}$ learning curve for each method on each problem. Each bar is normalized by TDRC's performance so that the errors for all problems can be shown in the same range. All results are averaged over 200 independent runs with standard error bars shown at the top of each rectangle, though most are vanishingly small. Off-policy TD and V-Trace both diverge on Baird's Counterexample, which is represented by the bars going off the top of the plot. HTD's bar is also off the plot due to its oscillating behavior.

In Boyan's chain, however, TDC performed worse than the other algorithms. TDRC, on the other hand, which regularizes $\mathbf{v}$, significantly improved learning in Boyan's chain. TD and HTD performed very well across all problems except for Baird's. Finally, V-trace—which uses a TD update with importance sampling ratios clipped at 1—performed slightly worse than TD due to the introduced bias.

The results reported here for TDC do not match previous results which indicate performance generally as good as TD (White & White, 2016). The reason for this discrepancy is that previous results carefully tuned the second step-size parameter $\eta\alpha$ for TDC. The need to tune η is part of the difficulty in using TDC. To better understand the role it is playing here, we include an additional result where we sweep η as well as α for TDC; for completeness, we also include this sweep for GTD2 and HTD. We sweep $\eta \in \{2^{-6}, 2^{-5}, \ldots, 2^5, 2^6\}$. This allows for $\eta\alpha$ that is very near zero as well as $\eta\alpha$ much larger than α. The theory for TDC suggests η should be larger than 1. The results in Figure 4.4, however, demonstrate that TDC almost always preferred the smallest η; but for very small η TDC effectively performs a TD update. By picking a small η, TDC essentially keeps $\mathbf{v}$ near zero—its initialization—and so removes the gradient correction term. TDC was therefore able to match TD by simply tuning a parameter so that it effectively *was* TD. Unfortunately, this is not a

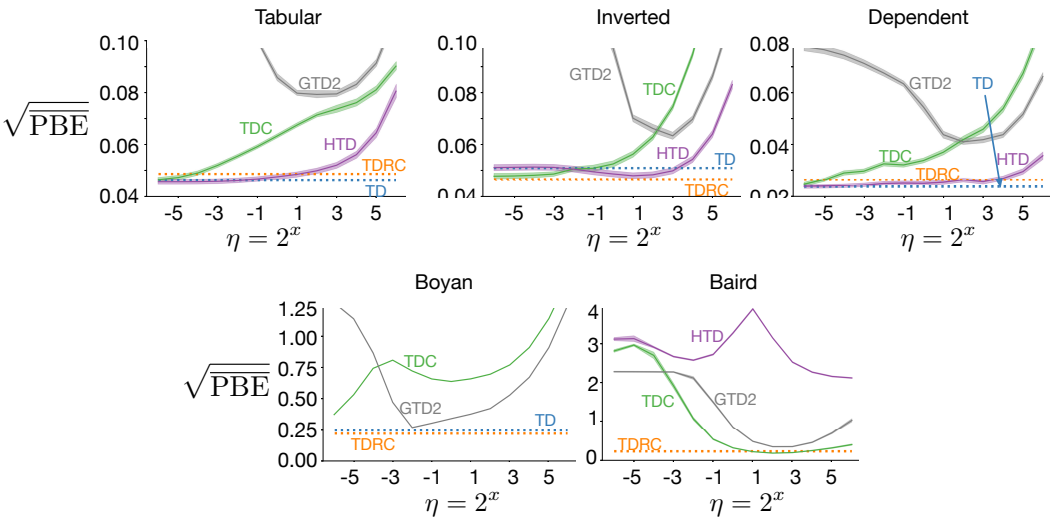


Figure 4.4: Sensitivity to the second step-size parameter, for changing parameter η. All methods used AdaGrad. All methods were free to choose any value of α for each η. Methods that do not have a second step-size parameter are shown as a flat line. Values swept are $\eta \in \{2^{-6}, 2^{-5}, \ldots, 2^5, 2^6\}$.

general strategy, for instance in Baird's, TDC picks $\eta \geq 1$ and small η performs poorly.

So far we have only used TDRC with a regularization parameter $\beta = 1$. This choice was made both to avoid over-tuning our method, as well as to show that an intuitive default value could be effective across settings. Intuitively, TDRC should not be sensitive to β, as both TDC ($\beta = 0$) and TD (large β) generally perform reasonably. Picking a $\beta > 0$ should enable TDRC to learn faster like TD—by providing a lower variance correction—as long as it's not too large, to ensure we avoid the divergence issues of TD.

We investigated this intuition by looking at performance across a range of $\beta \in 0.1 * \{2^0, 2^1, \ldots, 2^5, 2^6\}$. For $\beta = 0$, we have TDC. Ideally, performance should quickly improve for any non-negligible β, with a large flat region of good performance in the parameter sensitivity plots for a wide range of β. This is generally what we observe in Figure 4.5. For even very small β, TDRC noticeably improved performance over TDC, getting halfway between TDC and TD (Random Walk with Tabular or Dependent features) or in some cases immediately obtaining the good performance of TD (Random Walk with Inverted

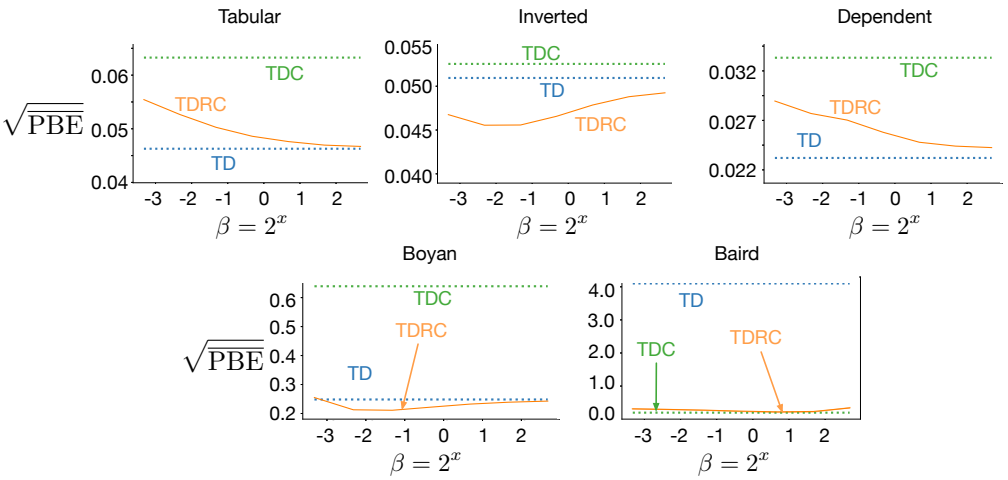


Figure 4.5: Sensitivity to the regularization parameter, β. TD and TDC are shown as dotted baselines, demonstrating extreme values of β; $\beta = 0$ represented by TDC and $\beta \to \infty$ represented by TD. This experiment demonstrates TDRC's notable insensitivity to β. Its similar range of values across problems, including Baird's counterexample, motivates that β can be chosen easily and is not heavily problem dependent. Values swept are: $\beta \in 0.1 * \{2^0, 2^1, \ldots, 2^5, 2^6\}$.

Features and Boyan's chain). Further, in these three cases, it even performed better or comparably to both TDC and TD for all tested β. Notably, these are the settings with more complex feature representations, suggesting that the regularization parameter helps TDRC learn an $\mathbf{v}$ that is less affected by harmful aliasing in the feature representation. Finally, the results also showed that $\beta = 1$ was in fact not optimal, and we could have obtained even better results in the previous section, typically with a larger β. These improvements, though, were relatively marginal over the choice of $\beta = 1$.

We provide the bar plot, sensitivity to the first step-size parameter, and sensitivity to the second step-size parameter in Figures 4.6, 4.7, and 4.8 when constant step-size parameters were used instead of AdaGrad. The conclusions remain similar to when AdaGrad was used to adapt a vector of step-size parameters.

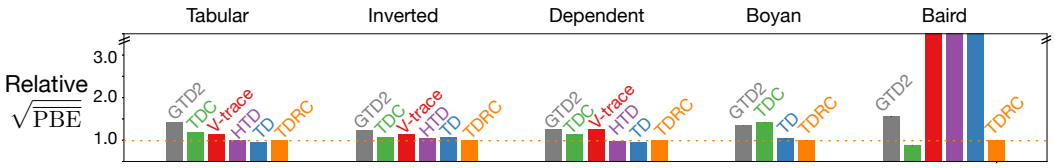


Figure 4.6: The normalized average area under the $\sqrt{\overline{\text{PBE}}}$ learning curve for each method on each problem using a constant step-size parameter. Each bar is normalized by TDRC's performance so that each problem can be shown in the same range. All results are averaged over 200 independent runs with standard error bars shown at the top of each rectangle, though most are vanishingly small.

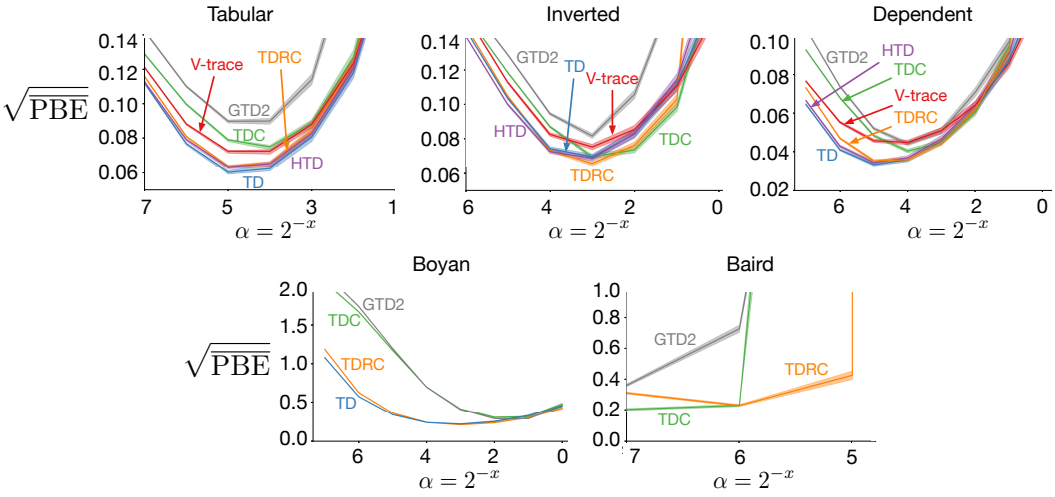


Figure 4.7: Step-size parameter sensitivity measured using average area under the $\sqrt{\overline{\text{PBE}}}$ learning curve. HTD and V-Trace are not shown in Boyan's Chain because they reduce to TD for on-policy problems.

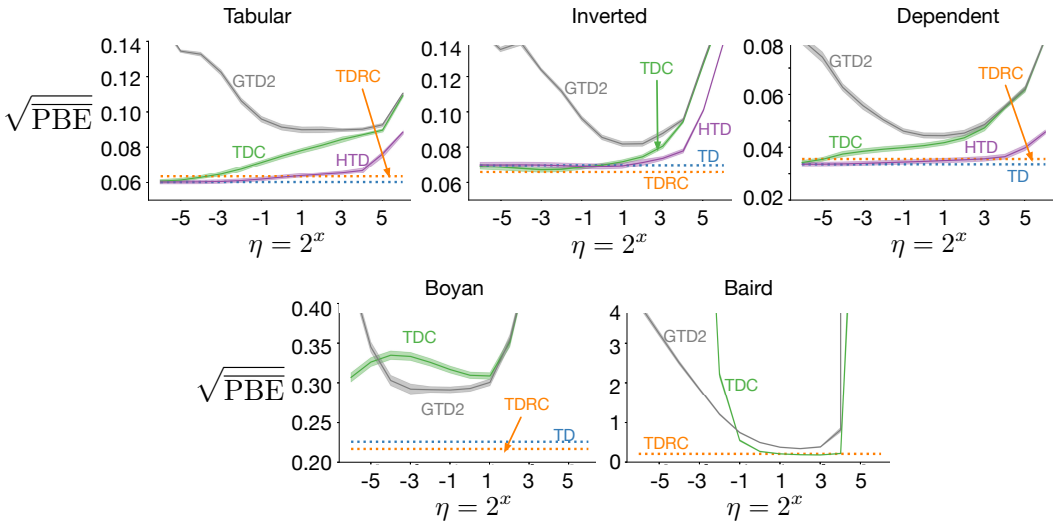


Figure 4.8: Sensitivity to the second step-size parameter, for changing parameter η. All methods use a constant step-size parameter α. All methods are free to choose any value of α for each specific value of η. Methods that do not have a second step-size parameter are shown as flat line.

4.4 Theoretically Characterizing the TDRC Update

The $\overline{\text{PBE}}$ (Sutton et al., 2009) is defined as

$$\begin{aligned}\overline{\text{PBE}}(\mathbf{w}_t) &\stackrel{\text{def}}{=} \mathbb{E}[\delta_t \mathbf{x}_t]^\top \mathbb{E}\left[\mathbf{x}_t \mathbf{x}_t^\top\right]^{-1} \mathbb{E}[\delta_t \mathbf{x}_t] \\ &= (-\mathbf{A}\mathbf{w} + \mathbf{b})^\top \mathbf{C}^{-1} (-\mathbf{A}\mathbf{w} + \mathbf{b})\end{aligned} \tag{4.3}$$

where $\mathbb{E}[\delta_t \mathbf{x}_t] = \mathbf{b} - \mathbf{A}\mathbf{w}_t$ for

$$\mathbf{C} \stackrel{\text{def}}{=} \mathbb{E}\left[\mathbf{x}_t \mathbf{x}_t^\top\right], \quad \mathbf{A} \stackrel{\text{def}}{=} \mathbb{E}\left[\mathbf{x}_t (\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top\right], \quad \mathbf{b} \stackrel{\text{def}}{=} \mathbb{E}[R_{t+1} \mathbf{x}_t].$$

The TD fixed point corresponds to $\mathbb{E}[\delta_t \mathbf{x}_t] = \mathbf{0}$ and so to the solution to the system $\mathbf{A}\mathbf{w}_t = \mathbf{b}$. The expectation is taken with respect to the target policy π, unless stated otherwise.

The expected update for TD corresponds to $\mathbb{E}[\delta_t \mathbf{x}_t] = \mathbf{b} - \mathbf{A}\mathbf{w}_t$. The expected update for $\mathbf{w}$ in TDC corresponds to the gradient of the $\overline{\text{PBE}}$,

$$-\frac{1}{2} \nabla \overline{\text{PBE}}(\mathbf{w}_t) = \mathbf{A}^\top \mathbf{C}^{-1} (\mathbf{b} - \mathbf{A}\mathbf{w}_t).$$

Both TDC and GTD2 estimate $\mathbf{v}_t \stackrel{\text{def}}{=} \mathbf{C}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{w}_t) = \mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^\top\right]^{-1}\mathbb{E}[\delta_t\mathbf{x}_t]$, to get the least squares estimate $\mathbf{v}_t^\top\mathbf{x}_t \approx \mathbb{E}[\delta_t|\mathbf{x}_t]$ for targets δ_t. TDC rearranges terms, to sample this gradient differently than GTD2; for a given $\mathbf{v}$, both have the same expected update for $\mathbf{w}$: $\mathbf{A}^\top\mathbf{v}$.

We can now consider the expected update for TDRC. Solving for the ℓ_2 regularized problem with target δ_t, we get $(\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^\top\right] + \beta\mathbf{I})\mathbf{v} = \mathbb{E}[\delta_t\mathbf{x}_t]$ which implies $\mathbf{v}_\beta = \mathbf{C}_\beta^{-1}(\mathbf{b} - \mathbf{A}\mathbf{w}_t)$ for $\mathbf{C}_\beta \stackrel{\text{def}}{=} \mathbf{C} + \beta\mathbf{I}$. To get a similar form to TDC, we consider the modified expected update $\mathbf{A}_\beta^\top\mathbf{v}_\beta$ for $\mathbf{A}_\beta \stackrel{\text{def}}{=} \mathbf{A} + \beta\mathbf{I}$. We can get the TDRC update by rearranging this expected update, similarly to how TDC is derived

$$\begin{aligned}
\mathbf{A}_\beta^\top\mathbf{v}_\beta &= (\mathbb{E}\left[(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})\mathbf{x}_t^\top\right] + \beta\mathbf{I})\mathbf{v}_\beta \\
&= \left(\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^\top\right] + \beta\mathbf{I} - \gamma\mathbb{E}\left[\mathbf{x}_{t+1}\mathbf{x}_t^\top\right]\right)\mathbf{C}_\beta^{-1}\mathbb{E}[\delta_t\mathbf{x}_t] \\
&= \left(\mathbb{E}\left[\mathbf{x}_t\mathbf{x}_t^\top\right] + \beta\mathbf{I}\right)\mathbf{C}_\beta^{-1}\mathbb{E}[\delta_t\mathbf{x}_t] - \gamma\mathbb{E}\left[\mathbf{x}_{t+1}\mathbf{x}_t^\top\right]\mathbf{C}_\beta^{-1}\mathbb{E}[\delta_t\mathbf{x}_t] \\
&= \mathbb{E}[\delta_t\mathbf{x}_t] - \gamma\mathbb{E}\left[\mathbf{x}_{t+1}\mathbf{x}_t^\top\right]\mathbf{v}_\beta.
\end{aligned}$$

This update equation for the primary weights looks precisely like the update in TDC, except that our $\mathbf{v}$ is estimated differently. Despite this difference, we show in Theorem 4.5.1 that the set of TDRC solutions $\mathbf{w}$ to $\mathbf{A}_\beta^\top\mathbf{v}_\beta = \mathbf{0}$ includes the TD fixed point, and this set is exactly equivalent if $\mathbf{A}_\beta$ is full rank.

In the following theorem we prove the convergence of TDRC. Though the TDRC updates are no longer gradients, we maintain the convergence properties of TDC. This theorem extends the TDC convergence result to allow for $\beta > 0$, where TDC corresponds to TDRC with $\beta = 0$. Theorem 4.4.1 shows that TDRC maintains convergence when TD is convergent: the case when $\mathbf{A}$ is positive definite. Otherwise, TDRC converges under more general settings than TDC, because it has the same conditions on η as given by Maei (2011) but allows for $\beta > 0$. The upper bound on β makes sense, since as $\beta \to \infty$, TDRC approaches TD.

Theorem 4.4.1 (Convergence of TDRC) *Consider the TDRC update, with*

a TDC like step-size parameter multiplier $\eta \geq 0$:

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \eta\alpha_t \left[\rho_t\delta_t - \mathbf{v}_t^\top \mathbf{x}_t\right] \mathbf{x}_t - \eta\alpha_t\beta\mathbf{v}_t, \tag{4.4}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t\rho_t\delta_t \mathbf{x}_t - \alpha_t\rho_t\gamma(\mathbf{v}_t^\top \mathbf{x}_t)\mathbf{x}_{t+1}, \tag{4.5}$$

*with step sizes $\alpha_t \in (0, 1]$, satisfying $\sum_{t=0}^{\infty}\alpha_t = \infty$ and $\sum_{t=0}^{\infty}\alpha_t^2 < \infty$. Assume that $(\mathbf{x}_t, R_t, \mathbf{x}_{t+1}, \rho_t)$ is an i.i.d. sequence with uniformly bounded second moments for states and rewards, $\mathbf{A} + \beta\mathbf{I}$ and $\mathbf{C}$ are non-singular, and that the standard coverage assumption (Sutton & Barto, 2018) holds, i.e. $b(A|S) > 0 \quad \forall S, A$ where $\pi(A|S) > 0$. Then $\mathbf{w}_t$ converges with probability one to the TD fixed point if **either** of the following are satisfied:*

*(i) $\mathbf{A}$ is positive definite, **or***

(ii) $\beta < -\lambda_{max}(\mathbf{H}^{-1}\mathbf{A}\mathbf{A}^\top)$ and $\eta > -\lambda_{min}(\mathbf{C}^{-1}\mathbf{H})$, with $\mathbf{H} \stackrel{def}{=} \frac{\mathbf{A}+\mathbf{A}^\top}{2}$. Note that when $\mathbf{A}$ is not positive definite, $-\lambda_{max}(\mathbf{H}^{-1}\mathbf{A}\mathbf{A}^\top)$ and $-\lambda_{min}(\mathbf{C}^{-1}\mathbf{H})$ are guaranteed to be positive real numbers.

Proof: We combine the TDRC update equations (Eqs. 4.4 and 4.5) into a single linear system in variable $\boldsymbol{\varrho}_t^\top \stackrel{\text{def}}{=} \left[\mathbf{v}_t^\top \mathbf{w}_t^\top\right]$:

$$\boldsymbol{\varrho}_{t+1} = \boldsymbol{\varrho}_t + \alpha_t(\mathbf{G}_{t+1}\boldsymbol{\varrho}_t + \mathbf{g}_{t+1}), \tag{4.6}$$

with $\mathbf{G}_{t+1} \stackrel{\text{def}}{=} \begin{bmatrix} -\eta(\mathbf{x}_t\mathbf{x}_t^\top + \beta\mathbf{I}) & \eta\rho_t\mathbf{x}_t(\gamma\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \\ -\rho_t(\gamma\mathbf{x}_{t+1}\mathbf{x}_t^\top) & \rho_t\mathbf{x}_t(\gamma\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \end{bmatrix}$ and $\mathbf{g}_{t+1} \stackrel{\text{def}}{=} \begin{bmatrix} \eta\rho_t R_{t+1}\mathbf{x}_t \\ \rho_t R_{t+1}\mathbf{x}_t \end{bmatrix}$.

For a random variable $\mathbf{X}$, using the definition of importance sampling, we know that $\mathbb{E}_b[\rho\mathbf{X}] = \mathbb{E}_\pi[\mathbf{X}]$. Further, while learning off–policy we assume the excursion setting and use the stationary state distribution corresponding to the behavior policy, i.e. $\mathbb{E}_\pi[\mathbf{x}_t\mathbf{x}_t^\top] = \sum_{S\in\mathcal{S}} d_b(S)\mathbf{x}(S)\mathbf{x}(S)^\top$, and consequently $\mathbb{E}_b[\mathbf{x}_t\mathbf{x}_t^\top] = \mathbb{E}_\pi[\mathbf{x}_t\mathbf{x}_t^\top]$. We define, $\mathbf{G} \stackrel{\text{def}}{=} \mathbb{E}_b[\mathbf{G}_k] = \begin{bmatrix} -\eta\mathbf{C}_\beta & -\eta\mathbf{A} \\ \mathbf{A}^\top - \mathbf{C} & -\mathbf{A} \end{bmatrix}$ and $\mathbf{g} \stackrel{\text{def}}{=} \mathbb{E}_b[\mathbf{g}_k] = \begin{bmatrix} \eta\mathbf{b} \\ \mathbf{b} \end{bmatrix}$, and therefore (4.6) can be rewritten as

$$\boldsymbol{\varrho}_{t+1} = \boldsymbol{\varrho}_t + \alpha_t\big(h(\boldsymbol{\varrho}_t) + M_{t+1}\big), \tag{4.7}$$

where $h(\boldsymbol{\varrho}) \stackrel{\text{def}}{=} \mathbf{G}\boldsymbol{\varrho} + \mathbf{g}$ and $M_{t+1} \stackrel{\text{def}}{=} (\mathbf{G}_{t+1} - \mathbf{G})\boldsymbol{\varrho}_t + (\mathbf{g}_{t+1} - \mathbf{g})$ is the noise sequence.

To prove the convergence of TDRC, we use the results from Borkar & Meyn (2000) which require the following to be true: (i) The function $h(\boldsymbol{\varrho})$

Box 2: Derivation of (4.8)

Following the analysis given in Maei (2011), we write

$$\det(\mathbf{G}-\lambda\,\mathbf{I}) = \det\begin{bmatrix} -\eta\,\mathbf{C}_\beta - \lambda\,\mathbf{I} & -\eta\,\mathbf{A} \\ \mathbf{A}^\top - \mathbf{C} & -\mathbf{A} - \lambda\,\mathbf{I}\end{bmatrix} = (-1)^{2d}\det\begin{bmatrix} \eta\,\mathbf{C}_\beta + \lambda\,\mathbf{I} & \eta\,\mathbf{A} \\ \mathbf{C} - \mathbf{A}^\top & \mathbf{A} + \lambda\,\mathbf{I}\end{bmatrix}.$$

For a matrix $\mathbf{U} = \begin{bmatrix}\mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{A}_3 & \mathbf{A}_4\end{bmatrix}$, $\det(\mathbf{U}) = \det(\mathbf{A}_1)\cdot\det(\mathbf{A}_4 - \mathbf{A}_3\,\mathbf{A}_1^{-1}\,\mathbf{A}_2)$. Further, since $\mathbf{C}$ is positive semi–definite, $\mathbf{C}_\beta + \lambda\,\mathbf{I}$ would be non–singular for any $\beta > 0$. Using these results, we get

$$\det(\mathbf{G}-\lambda\,\mathbf{I}) = \det(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I})\cdot\det(\mathbf{A} + \lambda\,\mathbf{I} - \eta(\mathbf{C}-\mathbf{A}^\top)(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I})^{-1}\,\mathbf{A}). \tag{B1}$$

Now $\eta\,\mathbf{C}\left(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I}\right)^{-1} = \Big(\big(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I}\big) - (\eta\beta+\lambda)\,\mathbf{I}\Big)\big(\eta\,\mathbf{C} + (\eta\beta + \lambda)\,\mathbf{I}\big)^{-1} = \mathbf{I} - (\eta\beta+\lambda)\big(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I}\big)^{-1}$. We can then write

$$\begin{aligned}
&\mathbf{A} + \lambda\,\mathbf{I} - \eta(\mathbf{C}-\mathbf{A}^\top)(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I})^{-1}\,\mathbf{A} \\
&= \mathbf{A} + \lambda\,\mathbf{I} - \eta\,\mathbf{C}(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I})^{-1}\,\mathbf{A} + \eta\,\mathbf{A}^\top(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I})^{-1}\,\mathbf{A} \\
&= \mathbf{A} + \lambda\,\mathbf{I} - \Big(\mathbf{I} - (\eta\beta+\lambda)\big(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I}\big)^{-1}\Big)\,\mathbf{A} + \eta\,\mathbf{A}^\top(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I})^{-1}\,\mathbf{A} \\
&= \lambda\,\mathbf{I} + (\eta\beta+\lambda)\big(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I}\big)^{-1}\,\mathbf{A} + \eta\,\mathbf{A}^\top(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I})^{-1}\,\mathbf{A} \\
&= \Big[\lambda\,(\mathbf{A})^{-1}\left(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I}\right) + (\eta\beta+\lambda)\,\mathbf{I} + \eta\,\mathbf{A}^\top\Big]\left(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I}\right)^{-1}\mathbf{A} \\
&= (\mathbf{A})^{-1}\Big[\lambda\big(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I}\big) + \mathbf{A}\left(\eta\,\mathbf{A}^\top + (\eta\beta+\lambda)\,\mathbf{I}\right)\Big]\big(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I}\big)^{-1}\,\mathbf{A}\,.
\end{aligned}$$

Putting the above result in (B1) along with the fact that $\det(\mathbf{A}_1\,\mathbf{A}_2) = \det(\mathbf{A}_1)\cdot\det(\mathbf{A}_2)$, we get

$$\det(\mathbf{G}-\lambda\,\mathbf{I}) = \det\Big(\lambda\big(\eta\,\mathbf{C} + (\eta\beta+\lambda)\,\mathbf{I}\big) + \mathbf{A}\left(\eta\,\mathbf{A}^\top + (\eta\beta+\lambda)\,\mathbf{I}\right)\Big).$$

is Lipschitz and there exists $h_\infty(\boldsymbol{\varrho}) \stackrel{\text{def}}{=} \lim_{c\to\infty}\frac{h(c\,\boldsymbol{\varrho})}{c}$ for all $\boldsymbol{\varrho} \in \mathbb{R}^{2d}$ (ii) The sequence $(M_t, \mathcal{F}_t)$ is a Martingale difference sequence (MDS), where $\mathcal{F}_t \stackrel{\text{def}}{=} \sigma(\boldsymbol{\varrho}_1, M_1, \ldots, \boldsymbol{\varrho}_t, M_t)$, and $\mathbb{E}[\|M_{t+1}\|^2 \mid \mathcal{F}_t] \le c_0(1 + \|\,\boldsymbol{\varrho}\,\|^2)$ for any initial parameter vector $\boldsymbol{\varrho}_1$ and some constant $c_0 > 0$; (iii) The step size sequence α_t satisfies $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$; (iv) The origin is a globally asymptotically stable equilibrium for the ODE $\dot{\boldsymbol{\varrho}} = h_\infty(\boldsymbol{\varrho})$; and (v) The ODE $\dot{\boldsymbol{\varrho}} = h(\boldsymbol{\varrho})$

has a unique globally asymptotically stable equilibrium.

The function $h(\boldsymbol{\varrho}) = \mathbf{G}\,\boldsymbol{\varrho} + \mathbf{g}$ is Lipschitz with the coefficient $\| \mathbf{G} \|$ and $h_\infty(\boldsymbol{\varrho}) = \mathbf{G}\,\boldsymbol{\varrho}$ is well defined for all $\boldsymbol{\varrho} \in \mathbb{R}^{2d}$. $(M_t, \mathcal{F}_t)$ is an MDS, since by construction it satisfies $\mathbb{E}[M_{t+1} \mid \mathcal{F}_t] = 0$ and $M_t \in \mathcal{F}_t$. The coverage assumption implies that the second moments of ρ_t are uniformly bounded. Then applying triangle inequality to $M_{t+1} = (\mathbf{G}_{t+1} - \mathbf{G})\,\boldsymbol{\varrho}_t + (\mathbf{g}_{t+1} - \mathbf{g})$ and using the boundedness of second moments of the quadruplets $(\mathbf{x}_t, R_t, \mathbf{x}_{t+1}, \rho_t)$, we get $\mathbb{E}[\|M_{t+1}\|^2 \mid \mathcal{F}_t] \leq \mathbb{E}[\|(\mathbf{G}_{t+1} - \mathbf{G})\,\boldsymbol{\varrho}_t\|^2 \mid \mathcal{F}_t] + \mathbb{E}\left[\| \mathbf{g}_{t+1} - \mathbf{g} \|^2 \mid \mathcal{F}_t\right] \leq c_0(\| \boldsymbol{\varrho}_t \|^2 + 1)$. Condition on the step size parameter follows from our assumptions in the theorem statement. To verify the conditions (iv) and (v), we first show that the real parts of all the eigenvalues of $\mathbf{G}$ are negative.

Proving that the Real Parts of Eigenvalues of G are Negative (assuming C to be non–Singular) We consider the case when the $\mathbf{C}$ matrix is non–singular. TDRC converges even when $\mathbf{C}$ is singular under alternate conditions, which are given in Section 4.4. From Box 2, we obtain

$$\det(\mathbf{G} - \lambda\,\mathbf{I}) = \det\Big(\lambda(\eta\,\mathbf{C} + (\eta\beta + \lambda)\,\mathbf{I}) + \mathbf{A}(\eta\,\mathbf{A}^\top + (\eta\beta + \lambda)\,\mathbf{I})\Big), \tag{4.8}$$

for some $\lambda \in \mathbb{C}$. Now because an eigenvalue λ of matrix $\mathbf{G}$ satisfies $\det(\mathbf{G} - \lambda\,\mathbf{I}) = 0$, there must exist a non–zero vector $\mathbf{z} \in \mathbb{C}^d$ such that $\mathbf{z}^*[\lambda(\eta\,\mathbf{C} + (\eta\beta + \lambda)\,\mathbf{I}) + \mathbf{A}(\eta\,\mathbf{A}^\top + (\eta\beta + \lambda)\,\mathbf{I})]\,\mathbf{z} = 0$, which is equivalent to

$$\lambda^2 + \left(\eta\beta + \eta\frac{\mathbf{z}^*\,\mathbf{C}\,\mathbf{z}}{\| \mathbf{z} \|^2} + \frac{\mathbf{z}^*\,\mathbf{A}\,\mathbf{z}}{\| \mathbf{z} \|^2}\right)\lambda + \eta\left(\beta\frac{\mathbf{z}^*\,\mathbf{A}\,\mathbf{z}}{\| \mathbf{z} \|^2} + \frac{\mathbf{z}^*\,\mathbf{A}\,\mathbf{A}^\top\,\mathbf{z}}{\| \mathbf{z} \|^2}\right) = 0.$$

We define $b_c = \frac{\mathbf{z}^*\,\mathbf{C}\,\mathbf{z}}{\| \mathbf{z} \|^2}$, $b_a = \frac{\mathbf{z}^*\,\mathbf{A}\,\mathbf{A}^\top\,\mathbf{z}}{\| \mathbf{z} \|^2}$, and $\lambda_z = \frac{\mathbf{z}^*\,\mathbf{A}\,\mathbf{z}}{\| \mathbf{z} \|^2} = \lambda_r + \lambda_c i$ for some $\lambda_r, \lambda_c \in \mathbb{R}$. The constants b_c and b_a are real and greater than zero for all non–zero vectors $\mathbf{z}$. Then the above equation can be written as

$$\lambda^2 + (\eta\beta + \eta b_c + \lambda_z)\,\lambda + \eta(\beta\lambda_z + b_a) = 0. \tag{4.9}$$

We solve for λ in Eq. 4.9 (see Box 3 for the full derivation) to obtain $2\lambda = -\Omega - \lambda_c i \pm \sqrt{(\Omega^2 - \Xi) + (2\Omega\lambda_c - 4\eta\beta\lambda_c)i}$, where we introduced intermediate variables $\Omega = \eta\beta + \eta b_c + \lambda_r$, and $\Xi = \lambda_c^2 + 4\eta(\beta\lambda_r + b_a)$, which are both real numbers.

Box 3: Solutions of (4.9)

The solutions of a quadratic $ax^2+bx+c=0$ are given by $x=-\frac{b}{2a}\pm\frac{\sqrt{b^2-4ac}}{2a}$. Using this, we solve for λ in Eq. 4.9:

$$\begin{aligned}
2\lambda &= -(\eta\beta+\eta b_c+\lambda_z)\pm\sqrt{(\eta\beta+\eta b_c+\lambda_z)^2-4\eta(\beta\lambda_z+b_a)}\\
&= -\big(\eta\beta+\eta b_c+(\lambda_r+\lambda_c i)\big)\pm\sqrt{\big(\eta\beta+\eta b_c+(\lambda_r+\lambda_c i)\big)^2-4\eta\big(\beta(\lambda_r+\lambda_c i)+b_a\big)}\\
&= -\Omega-\lambda_c i\pm\sqrt{(\Omega+\lambda_c i)^2-4\eta(\beta\lambda_r+b_a)-4\eta\beta\lambda_c i}\\
&= -\Omega-\lambda_c i\pm\sqrt{\big(\Omega^2-\lambda_c^2-4\eta(\beta\lambda_r+b_a)\big)+\big(2\Omega\lambda_c-4\eta\beta\lambda_c\big)i}\\
&= -\Omega-\lambda_c i\pm\sqrt{\big(\Omega^2-\Xi\big)+\big(2\Omega\lambda_c-4\eta\beta\lambda_c\big)i},
\end{aligned}$$

where in the second step we put $\lambda_z=\lambda_r+\lambda_c i$, and also we define $\Omega=\eta\beta+\eta b_c+\lambda_r$ and $\Xi=\lambda_c^2+4\eta(\beta\lambda_r+b_a)$, which are both real numbers.

Using $\mathrm{Re}(\sqrt{x+yi})=\pm\frac{1}{\sqrt{2}}\sqrt{\sqrt{x^2+y^2}+x}$ we get $\mathrm{Re}(2\lambda)=-\Omega\pm\frac{1}{\sqrt{2}}\sqrt{\Upsilon}$, with the intermediate variable $\Upsilon=\sqrt{(\Omega^2-\Xi)^2+(2\Omega\lambda_c-4\eta\beta\lambda_c)^2}+(\Omega^2-\Xi)$. Next we obtain conditions on β and η such that the real parts of both the values of λ are negative for all non–zero vectors $\mathbf{z}\in\mathbb{C}$.

CASE 1 First consider $\mathrm{Re}(2\lambda)=-\Omega+\frac{1}{\sqrt{2}}\sqrt{\Upsilon}$. Then $\mathrm{Re}(\lambda)<0$ is equivalent to

$$\Omega>\frac{1}{\sqrt{2}}\sqrt{\Upsilon}. \tag{4.10}$$

Since, the right hand side of this inequality is clearly positive, we must have

$$\Omega=\eta\beta+\eta b_c+\lambda_r>0. \tag{C1}$$

This gives us our first condition on η and β. Simplifying (4.10) and putting back the values for the intermediate variables (see Box 4 for details), we get

$$\Omega^2+\Xi>\sqrt{(\Omega^2-\Xi)^2+(2\Omega\lambda_c-4\eta\beta\lambda_c)^2}. \tag{4.11}$$

Again, since the right hand side of the above inequality is positive, we must have

$$\Omega^2+\Xi=(\eta\beta+\eta b_c+\lambda_r)^2+\lambda_c^2+4\eta(\beta\lambda_r+b_a)>0. \tag{C2}$$

Box 4: Simplification of (4.10)

Putting the value of $\Upsilon = \sqrt{(\Omega^2 - \Xi)^2 + (2\Omega\lambda_c - 4\eta\beta\lambda_c)^2} + (\Omega^2 - \Xi)$ back in $\Omega > \frac{1}{\sqrt{2}}\sqrt{\Upsilon}$, we get

$$\Omega > \frac{1}{\sqrt{2}}\sqrt{\sqrt{(\Omega^2 - \Xi)^2 + (2\Omega\lambda_c - 4\eta\beta\lambda_c)^2} + (\Omega^2 - \Xi)}$$

$$\Leftrightarrow \quad \Omega^2 > \frac{1}{2}\left[\sqrt{(\Omega^2 - \Xi)^2 + (2\Omega\lambda_c - 4\eta\beta\lambda_c)^2} + (\Omega^2 - \Xi)\right]$$

[squaring both sides]

$$\Leftrightarrow \quad \Omega^2 + \Xi > \sqrt{(\Omega^2 - \Xi)^2 + (2\Omega\lambda_c - 4\eta\beta\lambda_c)^2}$$

$$\Leftrightarrow \quad (\Omega^2 + \Xi)^2 > (\Omega^2 - \Xi)^2 + (2\Omega\lambda_c - 4\eta\beta\lambda_c)^2$$

[squaring both sides]

$$\Leftrightarrow \quad \Omega^2\Xi > (\Omega\lambda_c - 2\eta\beta\lambda_c)^2$$

$$\Leftrightarrow \quad \Omega^2(\lambda_c^2 + 4\eta(\beta\lambda_r + b_a)) > \Omega^2\lambda_c^2 + 4\eta^2\beta^2\lambda_c^2 - 4\eta\beta\lambda_c^2\Omega$$

[putting $\Xi = \lambda_c^2 + 4\eta(\beta\lambda_r + b_a)$]

$$\Leftrightarrow \quad \Omega^2\eta(\beta\lambda_r + b_a) > \eta^2\beta^2\lambda_c^2 - \eta\beta\lambda_c^2\Omega$$

$$\Leftrightarrow \quad (\eta\beta + \eta b_c + \lambda_r)^2(\beta\lambda_r + b_a) > \eta\beta^2\lambda_c^2 - \beta\lambda_c^2(\eta\beta + \eta b_c + \lambda_r)$$

[putting $\Omega = \eta\beta + \eta b_c + \lambda_r$]

$$\Leftrightarrow \quad (\eta\beta + \eta b_c + \lambda_r)^2(\beta\lambda_r + b_a) > -\beta\lambda_c^2(\eta b_c + \lambda_r)$$

$$\Leftrightarrow \quad (\eta\beta + \eta b_c + \lambda_r)^2(\beta\lambda_r + b_a) + \beta\lambda_c^2(\eta b_c + \lambda_r) > 0.$$

Note that all these steps have full equivalence (especially the squaring operations in second and fourth step are completely reversible), because we explicitly enforce that $\Omega > 0$ and $\Omega^2 + \Xi > 0$ in Conditions C1 and C2 respectively. As a result, if we satisfy conditions C1, C2, and C3, $\text{Re}(2\lambda) = -\Omega + \frac{1}{\sqrt{2}}\sqrt{\Upsilon} < 0$ would be satisfied as well.

This is the second condition we have on η and β. Continuing to simplify the inequality in (4.11) (again see Box 4 for details), we get our third and final condition:

$$(\eta\beta + \eta b_c + \lambda_r)^2(\beta\lambda_r + b_a) + \beta\lambda_c^2(\eta b_c + \lambda_r) > 0. \tag{C3}$$

If $\mathbf{A}$ is positive definite (in which case TD converges) then $\lambda_r > 0$ for all $\mathbf{z} \in \mathbb{R}$ and each of the Conditions C1, C2, and C3 hold true and consequently TDRC converges.

Now we show that TDRC converges even when $\mathbf{A}$ is not positive definite (the case where TD is not guaranteed to converge). If we assume $\beta\lambda_r + b_a > 0$ and $\eta b_c + \lambda_r > 0$, then each of the Conditions C1, C2, and C3 again hold true

and TDRC would converge. As a result we obtain the following bounds:

$$\beta < -\frac{b_a}{\lambda_r} \Rightarrow \beta < \min_{\mathbf{z}} \left(-\frac{\mathbf{z}^* \, \mathbf{A} \, \mathbf{A}^\top \, \mathbf{z}}{\mathbf{z}^* \, \mathbf{H} \, \mathbf{z}} \right), \tag{4.12}$$

$$\eta > -\frac{\lambda_r}{b_c} \Rightarrow \eta > \max_{\mathbf{z}} \left(-\frac{\mathbf{z}^* \, \mathbf{H} \, \mathbf{z}}{\mathbf{z}^* \, \mathbf{C} \, \mathbf{z}} \right), \tag{4.13}$$

with $\mathbf{H} \stackrel{\text{def}}{=} \frac{1}{2}(\mathbf{A} + \mathbf{A}^\top)$. These bounds can be made more interpretable. Using the substitution $\mathbf{y} = \mathbf{H}^{\frac{1}{2}} \, \mathbf{z}$ we obtain

$$\begin{aligned} \min_{\mathbf{z}} \left(-\frac{\mathbf{z}^* \, \mathbf{A} \, \mathbf{A}^\top \, \mathbf{z}}{\mathbf{z}^* \, \mathbf{H} \, \mathbf{z}} \right) &\equiv \min_{\mathbf{y}} \frac{\mathbf{y}^* (-\mathbf{H}^{-\frac{1}{2}} \, \mathbf{A} \, \mathbf{A}^\top \, \mathbf{H}^{-\frac{1}{2}}) \, \mathbf{y}}{\| \, \mathbf{y} \, \|^2} \\ &= \lambda_{\min}(-\mathbf{H}^{-\frac{1}{2}} \, \mathbf{A} \, \mathbf{A}^\top \, \mathbf{H}^{-\frac{1}{2}}) \\ &= -\lambda_{\max}(\mathbf{H}^{-\frac{1}{2}} \, \mathbf{A} \, \mathbf{A}^\top \, \mathbf{H}^{-\frac{1}{2}}) \\ &= -\lambda_{\max}(\mathbf{H}^{-1} \, \mathbf{A} \, \mathbf{A}^\top), \end{aligned}$$

where $\lambda_{\max}$ represents the maximum eigenvalue of the matrix. Proceeding similarly for η, we can write the bounds in Eq. 4.12 and 4.13 equivalently as

$$\beta < -\lambda_{\max}(\mathbf{H}^{-1} \, \mathbf{A} \, \mathbf{A}^\top), \tag{4.14}$$

$$\eta > -\lambda_{\min}(\mathbf{C}^{-1} \, \mathbf{H}). \tag{4.15}$$

If these bounds are satisfied by η and β then the real parts of all the eigenvalues of $\mathbf{G}$ would be negative and TDRC will converge.

CASE 2 Next consider $\text{Re}(2\lambda) = -\Omega - \frac{1}{\sqrt{2}}\sqrt{\Upsilon}$. The second term is always negative and we assumed $\Omega > 0$ in Eq. C1. As a result, $\text{Re}(\lambda) < 0$ and we are done.

Therefore, we get that the real part of the eigenvalues of $\mathbf{G}$ are negative and consequently condition (iv) above is satisfied. To show that condition (v) holds true, note that since we assumed $\mathbf{A} + \beta \, \mathbf{I}$ to be non–singular, $\mathbf{G}$ is also non–singular; this means that for the ODE $\dot{\boldsymbol{\varrho}} = h(\boldsymbol{\varrho})$, $\boldsymbol{\varrho}^* = -\mathbf{G}^{-1} \, \mathbf{g}$ is the unique asymptotically stable equilibrium with $\bar{\mathbf{V}}(\boldsymbol{\varrho}) \stackrel{\text{def}}{=} \frac{1}{2}(\mathbf{G} \, \boldsymbol{\varrho} + \mathbf{g})^\top (\mathbf{G} \, \boldsymbol{\varrho} + \mathbf{g})$ as its associated strict Lyapunov function.

We can extend this result to allow for singular $\mathbf{C}$, which was not possible for TDC. The set of conditions on η and β, however, are more complex. We include this result with conditions given in (4.16).

Convergence of TDRC when C is Singular When $\mathbf{C}$ is singular, $b_c = \frac{\mathbf{z}^* \mathbf{C} \mathbf{z}}{\| \mathbf{z} \|^2}$ is no longer always greater than zero for an arbitrary vector $\mathbf{z}$. Consequently, if we explicitly set $b_c = 0$ we would get alternative bounds on η and β for which TDRC would converge. Putting $b_c = 0$ in Conditions C1, C2, and C3, we get

$$\eta\beta + \lambda_r > 0,$$
$$(\eta\beta + \lambda_r)^2 + \lambda_c^2 + 4\eta(\beta\lambda_r + b_a) > 0, \text{and}$$
$$(\eta\beta + \lambda_r)^2(\beta\lambda_r + b_a) + \beta\lambda_c^2\lambda_r > 0.$$

As before, we are concerned with the case when $\mathbf{A}$ is not PSD and thus $\lambda_r < 0$. Further, assume that $\beta\lambda_r + b_a > 0$ (this is the same upper bound on β as given in Eq. 4.12). We simplify the third inequality above to obtain the bound on η. As a result, we get the following bounds for β and η:

$$\beta < -\frac{b_a}{\lambda_r}, \qquad \eta > \frac{1}{\beta}\left(\sqrt{\frac{-\beta\lambda_c^2\lambda_r}{\beta\lambda_r + b_a}} - \lambda_r\right). \tag{4.16}$$

The bound on η automatically satisfies the first condition $\eta\beta + \lambda_r > 0$. Therefore, if β and η satisfy these bounds, TDRC converges even for a singular $\mathbf{C}$ matrix. ■

4.5 Fixed Points of TDRC

Theorem 4.5.1 (Fixed Points of TDRC) *If* $\mathbf{w}$ *is a TD fixed point, i.e., a solution to* $\mathbf{A}\mathbf{w} = \mathbf{b}$*, then it is a fixed point for the expected TDRC update,*

$$\mathbf{A}_\beta^\top \mathbf{C}_\beta^{-1}(\mathbf{b} - \mathbf{A}\mathbf{w}) = \mathbf{0}.$$

Further, the set of fixed points for TD and TDRC are equivalent if $\mathbf{C}_\beta$ *is invertible and if* $-\beta$ *does not equal to any of the eigenvalues of* $\mathbf{A}$ *(so that* $\mathbf{A}_\beta$ *is non-singular). Note that* $\mathbf{C}_\beta$ *is always invertible if* $\beta > 0$*, and is invertible if* $\mathbf{C}$ *is invertible even for* $\beta = 0$*.*

Proof: To show equivalence, the first part is straightforward: when $\mathbf{A}\mathbf{w} = \mathbf{b}$, then $\mathbf{b} - \mathbf{A}\mathbf{w} = \mathbf{0}$ and so $\mathbf{A}_\beta^\top \mathbf{C}_\beta^{-1}(\mathbf{b} - \mathbf{A}\mathbf{w}) = \mathbf{0}$. This means that any TD

fixed point is a TDRC fixed point. Now we simply need to show that under the additional conditions, a TDRC fixed point is a TD fixed point.

If $-\beta$ does not equal any of the eigenvalues of $\mathbf{A}$, then $\mathbf{A}_\beta = \mathbf{A} + \beta\mathbf{I}$ is a full rank matrix. Because both $\mathbf{A}_\beta$ and $\mathbf{C}_\beta$ are full rank, $\mathbf{A}_\beta^\top \mathbf{C}_\beta^{-1}(\mathbf{b} - \mathbf{A}\mathbf{w})$ equals to $\mathbf{0}$ if and only if $\mathbf{b} - \mathbf{A}\mathbf{w} = \mathbf{0}$.

We can prove Theorem 4.5.1, in an alternate fashion as well. The linear system in Eq. 4.6 has a solution (in expectation) which satisfies

$$\mathbf{G}\,\boldsymbol{\varrho} + \mathbf{g} = \mathbf{0}.$$

We show that this linear system has full rank and thus a single solution: $\mathbf{w} = \mathbf{A}^{-1}\,\mathbf{b}$ and $\mathbf{v} = \mathbf{0}$. If we show that the matrix $\mathbf{G}$ is non–singular, i.e. its determinant is non–zero, we are done. From (4.8) it is straightforward to obtain

$$\det(\mathbf{G}) = \eta^{2d} \det(\mathbf{A}^\top + \beta\,\mathbf{I}) \cdot \det(\mathbf{A}),$$

which is non–zero if we assume that β does not equal the negative of any eigenvalue of $\mathbf{A}$ and that $\mathbf{A}$ is non–singular. ■

4.6 Conclusions

In this chapter, we introduced the TDRC algorithm: a simple modification of the TDC algorithm that achieves performance much closer to that of Off-policy TD. We established a standard way for setting the second step-size and regularization parameters. TDRC behaves like TD when TD performs well but also prevents divergence under off-policy sampling. TDRC is built on TDC, and, as we showed, inherits its soundness guarantees. In small linear prediction problems, TDRC performs best overall, when compared to many other prediction learning algorithms, and exhibits low sensitivity to its regularization parameter.

Chapter 5

Regularized Corrections for Control[1]

We set out to contribute two empirical studies and three algorithmic ideas in this dissertation. Two out of the three algorithmic ideas and both empirical studies are focused on prediction learning. One of the algorithmic ideas is intended for control learning, which is the focus of this chapter. This chapter includes the second contribution of this dissertation: a novel algorithm for off-policy control learning, which we call Q-learning with Regularized Corrections, or QRC for short. QRC is the control variant of the TDRC algorithm introduced in the previous chapter.

There are natural—though in some cases heuristic—extensions from prediction learning algorithms to both the control setting and to non-linear function approximation. In this chapter, we propose the QRC algorithm and investigate its practicality. We first investigate QRC in control with linear function approximation. We then provide a heuristic strategy to use TDRC and TDC with non-linear function approximation. We demonstrate—for the first time—that Gradient-TD methods can outperform Q-learning when using neural networks in two classic control domains and two visual domains.

[1]The contents of this chapter are based on a paper co-authored by this author (Ghiassian et al., 2020).

5.1 The QRC Algorithm: Extending TDRC to Control and to Non-linear Function Approximation

In this section we describe how to extend TDRC to control, and to non-linear function approximation. The extension to non-linear function approximation is also applicable in the prediction setting; we therefore begin there. We then discuss the extension to control which involves estimating action-values for the greedy policy.

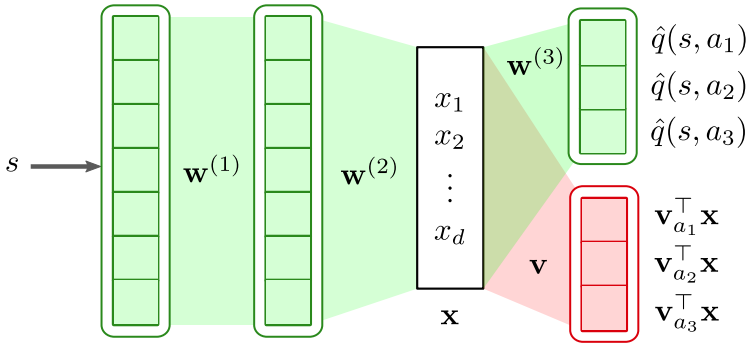


Figure 5.1: A schematic view of the network architecture in the non-linear function approximation setting. The gradients are only passed backward from the $\mathbf{w}$ head of the network. This choice is shown in the figure with the green color of the $\mathbf{w}$ head and the red color of the $\mathbf{v}$ head.

Consider the setting where we estimate $\hat{v}(s)$ using a neural network (Figure 5.1). The secondary weights in TDRC are used to obtain an estimate of $\mathbb{E}[\delta_t | S_t = s]$. Under linear function approximation, this expected TD error is estimated using linear regression with ℓ_2 regularization: $\mathbf{v}^\top \mathbf{x}_t \approx \mathbb{E}[\delta_t | S_t = s]$. With neural networks, this expected TD error can be estimated using an additional head on the network. The target for this second head is still δ_t, with a squared error and ℓ_2 regularization. One might even expect this estimate of $\mathbb{E}[\delta_t | S_t = s]$ to improve, when using a neural network, as compared to a hand-designed basis.

An important nuance is that gradients are not passed backward from the error in this second head. This choice is made for simplicity. The correction to the main weight vector is secondary, and we want to avoid degrading per-

formance in the value estimates simply to improve estimates of $\mathbb{E}[\delta_t|S_t = s]$. It also makes the connection to TD more clear when β becomes larger, as the update to the network is only impacted by $\mathbf{w}$. We have not extensively tested this choice; it remains to be seen if using gradients from both heads might actually be a better choice. For a schematic view of the architecture we used, see Figure 5.1.

The next step is to extend the algorithm to action-values. For an input state s, the network produces an estimate $\hat{q}(s, a)$ and a prediction $\hat{\delta}(s, a)$ of $\mathbb{E}[\delta_t|S_t = s, A_t = a]$ for each action. The weights $\mathbf{v}_{t+1,A_t}$ for the head corresponding to action A_t are updated using the features produced by the last layer, which we refer to as $\mathbf{x}_t$, with $\hat{\delta}(S_t, A_t) = \mathbf{v}_{t,A_t}^\top \mathbf{x}_t$:

$$\mathbf{v}_{t+1,A_t} \leftarrow \mathbf{v}_{t,A_t} + \alpha\big[\delta_t - \mathbf{v}_{t,A_t}^\top \mathbf{x}_t\big]\mathbf{x}_t - \alpha\beta\mathbf{v}_{t,A_t}. \tag{5.1}$$

For the other actions, the secondary weights are not updated since we did not get a target δ_t for them.

The remaining weights $\mathbf{w}_t$, which include all the weights in the network excluding $\mathbf{v}$, are updated using

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma\hat{q}(S_{t+1}, a') - \hat{q}(S_t, A_t) \\ \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha\delta_t\nabla_{\mathbf{w}}\hat{q}(S_t, A_t) - \alpha\gamma\hat{\delta}(S_t, A_t)\nabla_{\mathbf{w}}\hat{q}(S_{t+1}, a'), \end{aligned} \tag{5.2}$$

where a' is the action that the policy we are evaluating would take in state S_{t+1}. For control, we often select the greedy policy, and so $a' = \arg\max_a q(S_{t+1}, a)$ and (5.2) becomes $\delta_t = R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a) - \hat{q}(S_t, A_t)$ as in Q-learning. This action a' may differ from the (exploratory) action A_{t+1} that is actually executed, and so this estimation is off-policy.

We call this final algorithm QRC: Q-learning with Regularized Corrections. We can obtain, as a special case, a control algorithm based on TDC, which we call QC. QC is simply obtained by setting $\beta = 0$ in (5.1).

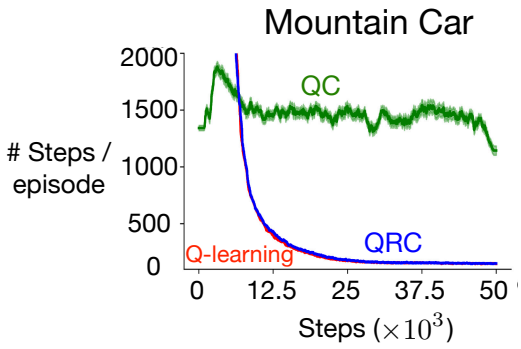


Figure 5.2: Number of steps to reach goal, averaged over runs, versus number of environment steps, in Mountain Car with tile-coded features. Comparison of state-action-value control algorithms with constant step-size parameters. Results are averaged over 200 independent runs, with shading corresponding to standard error. Q-learning and QRC found a good policy while QC failed to learn a policy that can reach the goal in reasonable time.

5.2 Control Experiments with Linear Function Approximation

We first tested the algorithms in a well-understood setting, in which we know Q-learning is effective: Mountain Car under a tile-coding representation.

In Mountain Car (Moore, 1990; Sutton, 1996), the goal is to reach the top of a hill with an underpowered car. The car starts at the bottom of the hill. The state consists of the agent's position and velocity, with a reward of -1 per step until termination, and actions to accelerate forward, backward or do nothing. The task is undiscounted.

We applied the algorithms to the Mountain Car control task, for 50,000 time steps and 200 independent runs. At the beginning of each of the runs, the weight vectors, $\mathbf{w}$ and $\mathbf{v}$, were set to $\mathbf{0}$. The behavior policy was ϵ-greedy with $\epsilon = 0.1$. Step-size parameter was swept over $\alpha \in \{2^{-8}, 2^{-7}, \ldots, 2^{-2}, 2^{-1}\}$ and then scaled by the number of active features. We used 16 tilings and 4×4 tiles. We fixed $\beta = 1$ for QRC, and $\eta = 1$ for QC.

The results are plotted in Figure 5.2. We plotted the best learning curve for each algorithm averaged over the 200 runs with standard error as shaded regions around each curve. We can see two clear outcomes from this con-

trol experiment. The control algorithm based on TDC failed to converge to a reasonable policy. The TDRC variant, on the other hand, matched the performance of Q-learning.

This result might be surprising, since the only difference between TDRC and TDC is regularizing $\mathbf{v}$. This small addition, though, seems to play a big role in avoiding this surprisingly bad performance of QC, and potentially explains why gradient methods have been dismissed as hard-to-use. When we looked more closely at QC's behavior, we found that the magnitude of the gradient corrections grew rapidly. This high magnitude gradient correction resulted in a higher magnitude gradient for $\mathbf{w}$, and pushed down the learning rate for QC. The constraint on this correction term provided by QRC seems to prevent this explosive growth, allowing QRC to attain comparable performance to the Q-learning agent.

5.3 Control Experiments with Non-linear Function Approximation

In the second phase of the control experiments, we used neural network function approximation in two classic control environments: Mountain Car and Cart Pole. In Cart Pole (Barto, Sutton & Anderson, 1983), the goal is to keep a pole balanced as long as possible, by moving a cart left or right. The state consists of the position and velocity of the cart, and the angle and angular velocity of the pole. The reward is +1 per step. Episodes end when the agent fails to balance the pole or balances the pole for more than 500 consecutive steps. The discount factor parameter, γ, is equal to 0.99.

We applied Q-learning, QC and QRC to the Mountain Car and Cart Pole tasks for 200 independent runs, where in Mountain Car each run lasted for 25,000 time steps and in Cart Pole each run lasted for 10,000 time steps. For the details of the experiment including the architecture and neural network parameters see Implementation Details, Section 5.5 of this chapter.

Learning curves are shown in the upper panels of Figure 5.3. QC learned more slowly than QRC and Q-learning in both environments. In the Mountain

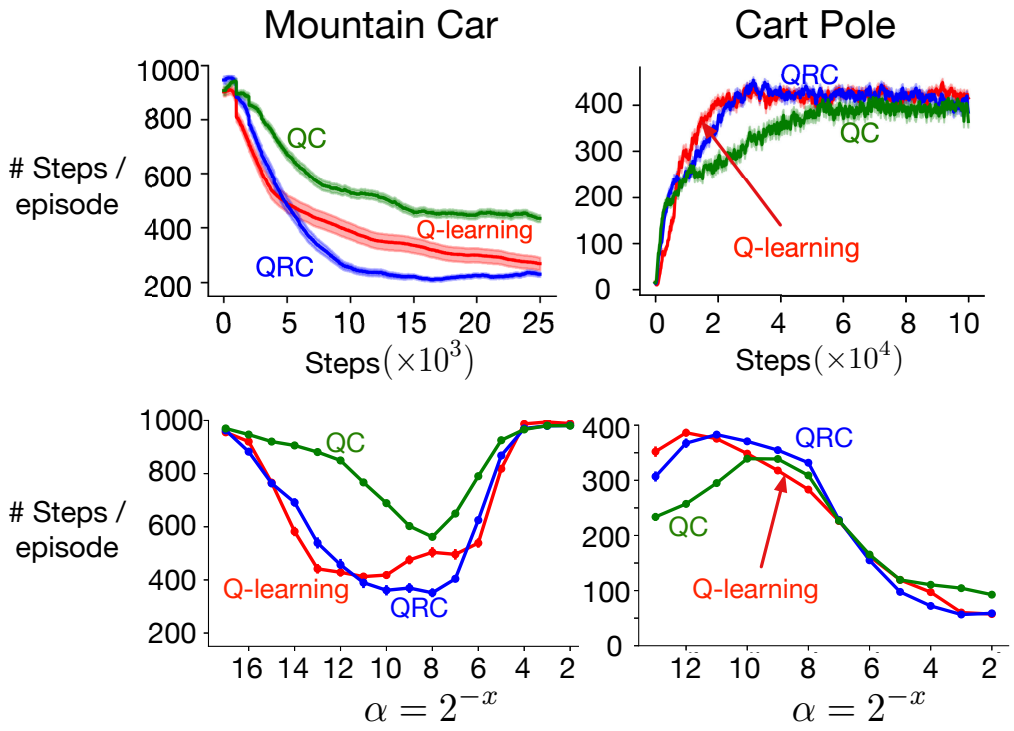


Figure 5.3: Performance of Q-learning, QC and QRC on two classic control environments with neural network function approximation. On top, the learning curves are shown. At the bottom, the parameter sensitivity for various step-size parameters are plotted. Lower is better for Mountain Car (fewer steps to goal) and higher is better for Cart Pole (more steps balancing the pole). Results are averaged over 200 runs, with shaded error corresponding to standard error. In Mountain Car QRC learned the fastest followed by Q-learning and QC. In Cart Pole Q-learning learned the fastest, followed by QRC, and then QC.

Car environment, QRC learned the fastest, followed by Q-learning and then QC.

5.4 Control Experiments with Non-linear Function Approximation in Visual Domains

To test the algorithms in more challenging visual domains, we used Breakout and Space Invaders from the MinAtar suite (Young & Tian, 2019). In Breakout, the agent moves a paddle left and right, to hit a ball into bricks. A reward of +1 is given for every brick hit; new rows appear when all the

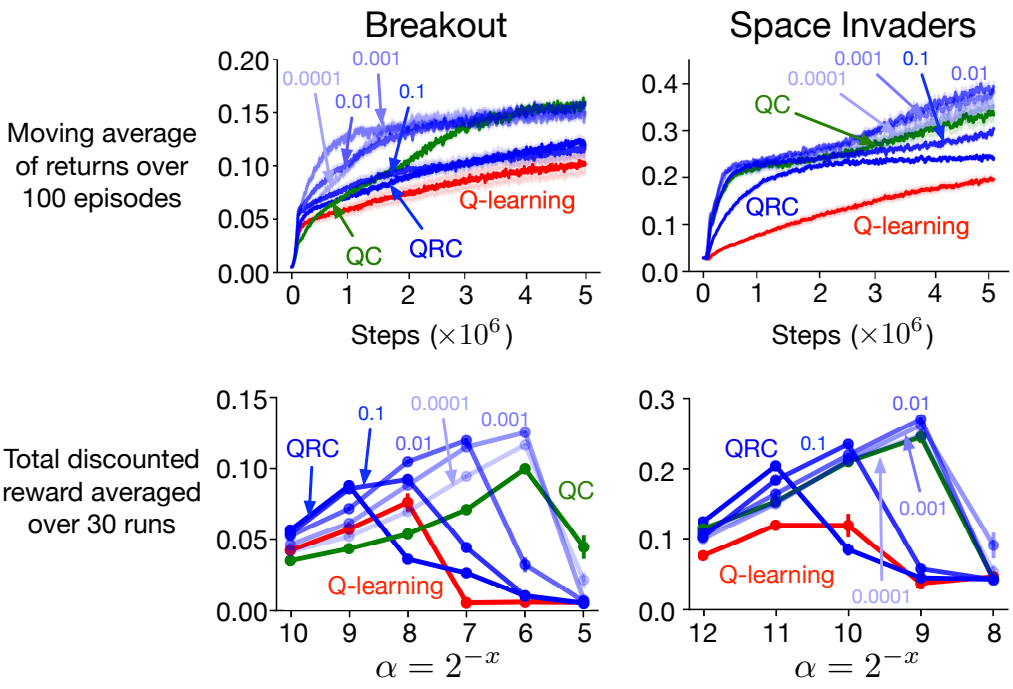


Figure 5.4: Performance of Q-learning, QC, and QRC in the two MinAtar environments. The learning curves in the top row depict the average return over time for the best performing step-size parameter for each agent. The step-size parameter sensitivity plots in the bottom row depict the total discounted reward achieved with several step-size parameter values. Higher is better. Results are averaged over 30 independent runs, with shaded error corresponding to standard error. Light blue lines show the performance of QRC with smaller regularization parameters, $\beta < 1$. QC provided a significant improvement on Q-learning. The best performance was achieved with QRC with $\beta < 1$.

rows are cleared. The episode ends when the agent misses the ball. In Space Invaders, the agent shoots alien ships coming towards it, and dodges their fire. A reward of +1 is given for every alien that is shot. The episode ends when the spaceship is hit by alien fire or reached by an alien ship. These environments are simplified versions from the Atari suite (Bellemare et al., 2013), designed to avoid the need for large networks and make it more feasible to complete more exhaustive comparison—similar to the ones conducted here—including using more runs.

We applied QRC, QC and Q-learning to the two MinAtar environments. Each algorithm instance was run for 30 independent runs, each of which lasted

for 5 million time steps. All methods use a network with one convolutional layer, followed by a fully connected layer. All experimental settings are identical to the original MinAtar paper (Young & Tian, 2019). The discount factor was $\gamma = 0.99$. For the details of the experiment, including the network architecture and the parameters, see the Implementation Details, Section 5.5 of this chapter.

The learning curve for the algorithm instance that resulted in the largest area under the learning curve is shown in the two upper panels of Figure 5.4. Each point in the learning curve is the moving average of returns over 100 episodes similar to Young and Tian (2019). To plot the sensitivity over parameters, we used the average of all returns acquired during the 5 million steps, and divided the results by the number of returns. See the bottom row of Figure 5.4. The code for this process is provided by Young and Tian at `https://github.com/kenjyoung/MinAtar`.

On the two MinAtar environments we obtained a surprising result: QC provided substantial performance improvements over Q-learning (see Figure 5.4). QRC with $\beta = 1$ was not as performant as QC in this setting and instead obtained performance in-between QC and Q-learning. However, QRC with smaller values of regularization parameter (shown as lighter blue lines) resulted in the best performance.

This outcome highlights that Gradient-TD methods are not only theoretically appealing, but could actually be a better alternative to Q-learning in standard (non-adversarially chosen) problems. It further shows that, although QRC with $\beta = 1$ generally provides a reasonable strategy, substantial improvements might be obtained with an adaptive method for selecting β.

5.5 Implementation Details

In this section, we provide the details of experiments that used non-linear function approximation.

Mountain Car and Cart Pole with Non-linear Function Approximation We applied Q-learning, QC and QRC to the Mountain Car and Cart Pole tasks for 200 independent runs, where in Mountain Car each run lasted for 25,000 time steps and in Cart Pole each run lasted for 10,000 time steps. To solve these task we used a fully connected neural network with two hidden layers where each layer had 64 nodes in Cart Pole (32 nodes in Mountain Car) with ReLU as the non–linearity and the output layer as linear. The weights were updated using a replay buffer of size 4,096 in Cart Pole and a replay buffer of size 4,000 in Mountain Car. In Cart Pole, we used a mini-batch size of 32 for updates and used Adam with $\epsilon = 10^{-8}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. In Mountain Car, we tested a more highly off-policy setting: 10 replay steps per time step. By using more replay per step, more data from older policies is used, resulting in a more off-policy data distribution. We used Adam to update the $\mathbf{v}$ vector using $\epsilon = 10^{-8}$, $\beta_1 = 0.99$, and $\beta_2 = 0.999$. The neural network weights for both tasks were initialized using Xavier initialization (Glorot & Bengio, 2010) and the biases were initialized with a normal distribution with mean 0 and standard deviation 0.1. The second weight vectors were initialized to with zero vectors. In both tasks, actions were selected using an ϵ-greedy policy with $\epsilon = 0.1$. We applied a range of algorithm instances with various step-size parameters to the problems: $\{2^{-13}, ..., 2^{-2}\}$ for Cart Pole and $\{2^{-17}, ..., 2^{-2}\}$ for Mountain Car. In these tasks we set $\eta = 1$ for QC and set the regularization parameter $\beta = 1$ for QRC. We did not use target networks in any of the tasks.

To plot the learning curves for the control experiments, we followed a simple procedure. During the experiment, we saved the number of steps per episode. For example, in the non-linear Mountain Car experiment, we saved the number of steps per episode over the 25,000 time steps of each run. Then, for each run, we created an array with 25,000 (or any other number of steps each run lasted for) elements. We filled the array with the number of steps each episode lasted. For example, if the first episode lasted 100 steps, and the second episode lasted 80 steps, the first 100 elements of the array were filled with the number 100, and the next 80 elements in the array were filled

with the number 80. Then, for each run, we smoothed the resulting array by averaging over a window size of 10. We then computed the average and standard deviation over the smoothed data, over runs. We finally plotted the learning curve of the algorithm instances that had the smallest (or largest for Cart Pole) area under the learning curve.

Breakout and Space Invaders with Non-linear Function Approximation We used a decayed ϵ-greedy policy with $\epsilon = 1$ decaying linearly to $\epsilon = 0.1$ over the first 100,000 steps. The rewards were scaled by $(R \times (1 - \gamma))$ so that the neural network does not have to estimate large returns. The Q-Learning and QRC network architectures were the same as that used by Young and Tian, (2019). The network had one convolutional layer and one fully connected layer after that. The convolutional layer used sixteen 3×3 convolutions with stride one. The fully connected layer had 128 units. Both convolutional and fully connected layers used ReLU gates. The network was initialized the same way as Young and Tian (2019). We did not use target networks for MinAtar experiments because Young and Tian (2019) showed that using target networks has negligible effects on the results. We used a circular replay buffer of size 100,000. The agent started learning when the replay buffer had 5,000 samples in it. The agent had one training step using a mini-batch of size 32 per environment step. As explained by Young and Tian (2019), frame skipping was not necessary since the frames of the MinAtar environment are more information rich. Other parameters were chosen the same as Young and Tian (2019): RMSProp with a smoothing constant of 0.95, and $\epsilon = 0.01$ was used. For QRC and QC, we used RMSProp to learn the second weight vector $\mathbf{v}$ as well. We swept over the RMSprop step-size parameter in powers of 2: $\{2^{-10}, ..., 2^{-5}\}$ for breakout, and $\{2^{-12}, ..., 2^{-8}\}$ for space invaders. The parameter η was set to 1 for QC and QRC and β was 1 for QRC.

5.6 Conclusions

We proposed QRC, the control variant of the TDRC algorithm, with extensions to non-linear function approximation. Our experiments suggest that the performance of QRC is comparable to, and in some cases notably better than, Q-learning. This constitutes the first demonstration of Gradient-TD methods outperforming Q-learning in visual domains. Our results suggest that this simple modification to the standard Q-learning update—i.e., QRC—could provide a more general purpose algorithm.

Chapter 6

An Empirical Comparison on the Collision Task[1]

One of the objectives of this dissertation is to provide a better understanding of online off-policy prediction learning algorithms and how they perform in practice. We take the first step in gaining a better understanding of the algorithms' merits and interrelationships in this chapter by presenting one of the most important contributions of this dissertation: a detailed empirical study of off-policy prediction learning algorithms on a small task, called the Collision task. In previous chapters, we discussed 10 of the existing off-policy prediction learning algorithms, and introduced one new prediction learning algorithm (TDRC). This chapter makes a comparison of all these algorithms.

With many algorithms proposed for off-policy prediction learning, it is important to know how these algorithms compare to each other in practice, in terms of learning speed, asymptotic error level, and ease of use. As discussed previously, high variance and divergence are arguably the two most important challenges of off-policy learning. Lots of the online off-policy prediction literature since the 2000s have focused on proposing algorithms that are guaranteed to converge under off-policy training with linear function approximation. There have been few studies that focused exclusively on the empirical considerations of off-policy prediction learning. To the best of our knowledge, three studies to date conducted detailed empirical comparisons of fully incremental

[1]The contents of this chapter are based on a paper co-authored by this author (Ghiassian & Sutton, 2021a).

off-policy prediction learning algorithms with linear function approximation: Geist and Scherrer (2014), Dann, Neumann, and Peters (2014), and White and White (2016). While these empirical studies played an important role in understanding the advantages and disadvantages of algorithms, our understanding of the algorithms merits remains limited.

In previous chapters we discussed the prediction variant of the Retrace algorithm, called V-trace, but postponed the derivation of the prediction variants of Tree Backup and ABQ to this chapter. To be able to apply ABQ and Tree Backup to prediction learning problems, we need to derive their prediction variants. In this chapter we derive the prediction variants of ABQ, Tree Backup, and Retrace algorithms in a unified way. To the best of our knowledge, this dissertation is the first to derive prediction variants of ABQ and Tree Backup. We refer to the prediction variant of ABQ as ABTD, and refer to the prediction and control variants of the Tree Backup algorithm with the same name.

This chapter presents empirical results with eleven prominent off-policy prediction learning algorithms that use linear function approximation: five Gradient-TD methods, two Emphatic-TD methods, Off-policy TD(λ), V-trace, Tree Backup, and ABTD. Our experiment used the Collision task, a small idealized off-policy problem analogous to that of an autonomous car trying to predict whether it will collide with an obstacle. We assessed the performance of the algorithms according to their learning rate, asymptotic error level, and sensitivity to step-size and bootstrapping parameters. By these measures, we partially order the eleven algorithms into three tiers on the Collision task. The top tier comprised of the two Emphatic-TD algorithms. These algorithms learned the fastest, reached the lowest error levels, and were most robust to parameter settings. The middle tier comprised of six Gradient-TD algorithms and Off-policy TD(λ). These algorithms were more sensitive to the bootstrapping parameter than the Emphatic-TD algorithms and did not learn faster than them. Finally, the bottom tier comprised V-trace, Tree Backup, and ABTD. These three algorithms were no faster and had higher asymptotic error than the others.

Our results are definitive for this task, though of course experiments with more tasks are needed before an overall assessment of the algorithms' merits can be made.

6.1 Two Different Forms of Importance Sampling Placement for Off-policy TD(λ)

The goal of this chapter is to conduct an empirical comparison of prediction learning algorithms. However, two of the algorithms that we intend to include in our comparison (Tree Backup and ABTD), are originally proposed for control. To derive these algorithms for prediction, and to show how these algorithms are related to the classic Off-policy TD(λ), we first need to show the Off-policy TD(λ) update rules can be written in two different forms. We show that these two sets of update rules are equivalent to each other step by step. One of these forms is the one that we previously used to introduce Off-policy TD(λ) in Chapter 2, and the other one is later used to derive the prediction variants of the three algorithms (Tree Backup, ABTD, and V-trace).

The original work on Off-policy TD(λ) uses the following update rules (Precup, Sutton, & Singh, 2000):

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t^\rho &\leftarrow \rho_t \left(\gamma\lambda \mathbf{z}_{t-1}^\rho + \mathbf{x}_t\right) \quad \text{with } \mathbf{z}_{-1}^\rho = \mathbf{0} \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t.
\end{aligned}$$

Other works (e.g., Geist & Scherrer, 2014) have used a different placement of the importance sampling ratio, with ρ_t in the update rule for $\mathbf{w}$ and ρ_{t-1} in the update rule for $\mathbf{z}$:

$$\begin{aligned}
\mathbf{z}_t' &\leftarrow \rho_{t-1}\gamma\lambda \mathbf{z}_{t-1}' + \mathbf{x}_t \quad \text{with } \mathbf{z}_{-1}' = \mathbf{0} \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \rho_t \delta_t \mathbf{z}_t'.
\end{aligned}$$

In Box 1, we show that given $\mathbf{z}_{-1}^\rho = \mathbf{z}_{-1}' = 0$, the two sets of updates listed above for Off-policy TD(λ) are equivalent step by step.

Box 1: Equality of the two forms of importance sampling placement

We start by showing that $\delta_t \mathbf{z}_t$ is equal to the product $\rho_t \delta_t \mathbf{z}'_t$ at each step, given that $\mathbf{z}^{\rho}_{-1} = \mathbf{z}'_{-1} = \mathbf{0}$.

$$\begin{aligned}
\rho_t \delta_t \mathbf{z}'_t &= \rho_t \delta_t \left[\rho_{t-1}\gamma\lambda \mathbf{z}'_{t-1} + \mathbf{x}_t\right] \\
&= \rho_t \delta_t \left[\rho_{t-1}\gamma\lambda(\rho_{t-2}\gamma\lambda \mathbf{z}'_{t-2} + \mathbf{x}_{t-1}) + \mathbf{x}_t\right] \\
&= \rho_t \delta_t \left[(\gamma\lambda)^2 \rho_{t-1}\rho_{t-2}\mathbf{z}'_{t-2} + \gamma\lambda\rho_{t-1}\mathbf{x}_{t-1} + \mathbf{x}_t\right] \\
&= \rho_t \delta_t \left[(\gamma\lambda)^2 \rho_{t-1}\rho_{t-2}\left(\rho_{t-3}\gamma\lambda \mathbf{z}'_{t-3} + \mathbf{x}_{t-2}\right) + \gamma\lambda\rho_{t-1}\mathbf{x}_{t-1} + \mathbf{x}_t\right] \\
&= \rho_t \delta_t \left[(\gamma\lambda)^3 \rho_{t-1}\rho_{t-2}\rho_{t-3}\mathbf{z}'_{t-3} + (\gamma\lambda)^2 \rho_{t-1}\rho_{t-2}\mathbf{x}_{t-2} + \gamma\lambda\rho_{t-1}\mathbf{x}_{t-1} + \mathbf{x}_t\right] \\
&\vdots \\
&= \rho_t \delta_t \left(\sum_{i=0}^{t} (\gamma\lambda)^i \mathbf{x}_{t-i} \prod_{k=1}^{i} \rho_{t-k}\right) + \rho_t \delta_t \left((\gamma\lambda)^{t+1} \prod_{k=1}^{t+1} \rho_{t-k}\right) \mathbf{z}'_{-1},
\end{aligned}$$

assuming that $\mathbf{z}'_{-1} = \mathbf{0}$:

$$\rho_t \delta_t \mathbf{z}'_t = \rho_t \delta_t \left(\sum_{i=0}^{t} (\gamma\lambda)^i \prod_{k=1}^{i} \rho_{t-k}\right). \tag{6.1}$$

On the other hand we have:

$$\begin{aligned}
\delta_t \mathbf{z}^{\rho}_t &= \delta_t \left[\rho_t \left(\gamma\lambda \mathbf{z}^{\rho}_{t-1} + \mathbf{x}_t\right)\right] \\
&= \rho_t \delta_t \left[\gamma\lambda\rho_{t-1} \left(\gamma\lambda \mathbf{z}^{\rho}_{t-2} + \mathbf{x}_{t-1}\right) + \mathbf{x}_t\right] \\
&= \rho_t \delta_t \left[(\gamma\lambda)^2 \rho_{t-1}\mathbf{z}^{\rho}_{t-2} + \gamma\lambda\rho_{t-1}\mathbf{x}_{t-1} + \mathbf{x}_t\right] \\
&= \rho_t \delta_t \left[(\gamma\lambda)^2 \rho_{t-1}\left(\rho_{t-2}\left(\gamma\lambda \mathbf{z}^{\rho}_{t-3} + \mathbf{x}_{t-2}\right)\right) + \gamma\lambda\rho_{t-1}\mathbf{x}_{t-1} + \mathbf{x}_t\right] \\
&= \rho_t \delta_t \left[(\gamma\lambda)^3 \rho_{t-1}\rho_{t-2}\mathbf{z}^{\rho}_{t-3} + (\gamma\lambda)^2 \rho_{t-1}\rho_{t-2}\mathbf{x}_{t-2} + \gamma\lambda\rho_{t-1}\mathbf{x}_{t-1} + \mathbf{x}_t\right] \\
&\vdots \\
&= \rho_t \delta_t \left(\sum_{i=0}^{t} (\gamma\lambda)^i \mathbf{x}_{t-i} \prod_{k=1}^{i} \rho_{t-k}\right) + \rho_t \delta_t \left((\gamma\lambda)^{t+1} \prod_{k=1}^{t} \rho_{t-k}\right) \mathbf{z}^{\rho}_{-1}
\end{aligned}$$

which is equal to (6.1) given that $\mathbf{z}^{\rho}_{-1} = \mathbf{0}$.

6.2 Derivations of Tree Backup, V-trace, and ABTD

In this section, we derive the prediction variants of Tree Backup(λ), Retrace(λ), and ABQ(ζ). Deriving the prediction variant of control algorithms is typically straightforward. However, deriving the prediction variant of the three mentioned algorithms is a little more involved. All prediction algorithms can be seen as Off-policy TD(λ) with λ_t generalized from a constant to a function of (S_t, A_t). As we will shortly see, importance sampling ratios cannot be completely avoided in the prediction setting as was done in the control setting. Trying to avoid all importance sampling ratios in the prediction learning case might result in an incorrect version of these algorithms that we will discuss at the end of this section.

As mentioned, the key idea is to set $\lambda_t = \lambda(S_{t-1}, A_{t-1})$ adaptively in generic Off-policy TD(λ):

$$\mathbf{z}_t = \rho_{t-1}\gamma_t\lambda_t\mathbf{z}_{t-1} + \mathbf{x}_t \tag{6.2}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\rho_t\delta_t\mathbf{z}_t, \tag{6.3}$$

where δ_t is the TD-error, $\mathbf{z}_t$ is eligibility trace, ρ_t is the importance sampling ratio, and α is the step-size parameter. The prediction variant of all three algorithms can be derived in a similar way. To understand the prediction variant of these algorithms, we derive ABTD(ζ). We then use ABTD(ζ) to derive extensions to V-trace(λ) and Tree Backup(λ) for prediction.

Consider the generalized λ-return, for a λ based on the state and action—as in ABQ(ζ)—or the entire transition (White, 2017). Let $\lambda_{t+1} = \lambda(S_t, A_t, S_{t+1})$ be defined based on the transition (S_t, A_t, S_{t+1}), corresponding to how rewards and discounts are defined based on the transition, $R_{t+1} = r(S_t, A_t, S_{t+1})$ and $\gamma_{t+1} = \gamma(S_t, A_t, S_{t+1})$. Then, given a value function $\hat{v}$, the λ-return G_t^λ for generalized γ and λ is defined recursively as

$$G_t^\lambda \stackrel{\text{def}}{=} \rho_t \left(R_{t+1} + \gamma_{t+1}\left[(1-\lambda_{t+1})\hat{v}(S_{t+1}) + \lambda_{t+1}G_{t+1}^\lambda\right]\right).$$

Similar to ABQ(ζ) (Mahmood et al., 2017, Equation 7), this λ-return can be

written using TD-errors

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

as

$$\begin{aligned}
G_t^\lambda &= \rho_t \left(R_{t+1} + \gamma_{t+1}\hat{v}(S_{t+1}) - \gamma_{t+1}\lambda_{t+1}\hat{v}(S_{t+1}) + \gamma_{t+1}\lambda_{t+1}G_{t+1}^\lambda\right) \\
&= \rho_t \left(\delta_t + \hat{v}(S_t) + \gamma_{t+1}\lambda_{t+1}\left[G_{t+1}^\lambda - \hat{v}(S_{t+1})\right]\right) \\
&= \rho_t\delta_t + \rho_t\hat{v}(S_t) + \\
&\rho_t\gamma_{t+1}\lambda_{t+1}\left(\rho_{t+1}\delta_{t+1} + \rho_{t+1}\gamma_{t+2}\lambda_{t+2}\left[G_{t+2}^\lambda - \hat{v}(S_{t+2})\right]\right) \\
&= \rho_t \sum_{n=t}^{\infty} (\rho_{t+1}\lambda_{t+1}\gamma_{t+1})^n \delta_t + \rho_t\hat{v}(S_t),
\end{aligned}$$

where

$$(\rho_{t+1}\lambda_{t+1}\gamma_{t+1})^n \stackrel{\text{def}}{=} \prod_{i=t+1}^{n} \rho_i\lambda_i\gamma_i.$$

This return differs from the return used by ABQ(ζ), because it corresponds to the return from a state, rather than the return from a state and action. In ABQ(ζ), the goal is to estimate the action-value for a given state and action. For ABTD(ζ), the goal is to estimate the value for a given state. For the return from a state S_t, we need to correct the distribution over actions A_t with importance sampling ratio ρ_t. For ABQ(ζ), the correction with ρ_t is not necessary because S_t and A_t are both given, and importance sampling corrections only need to be computed for future states and actions, with ρ_{t+1} onward. For ABTD(ζ), therefore, unlike ABQ(ζ), not all importance sampling ratios can be avoided. We can, however, still set λ in a similar way to ABQ(ζ) to mitigate the variance effects of importance sampling.

To ensure $\rho_t\lambda_{t+1}$ is well-behaved, ABTD(ζ) sets λ as follows:

$$\lambda(S_t, A_t, S_{t+1}) = \nu(\psi, S_t, A_t)b(S_t, A_t),$$

with the following scalar parameters to define ν_t (Mahmood, Yu, & Sutton,

2017):

$$\begin{aligned}
\nu_t &\stackrel{\text{def}}{=} \nu(\psi(\zeta), S_t, A_t) \stackrel{\text{def}}{=} \min\left(\psi(\zeta), \frac{1}{\max(b(A_t|S_t), \pi(A_t|S_t))}\right), \\
\psi(\zeta) &\stackrel{\text{def}}{=} 2\zeta\psi_0 + \max(0, 2\zeta - 1)(\psi_{\max} - 2\psi_0), \\
\psi_0 &\stackrel{\text{def}}{=} \frac{1}{\max_{s,a} \max(b(a|s), \pi(a|s))}, \\
\psi_{\max} &\stackrel{\text{def}}{=} \frac{1}{\min_{s,a} \max(b(a|s), \pi(a|s))}.
\end{aligned}$$

In the λ-return, then

$$\rho_t \lambda_{t+1} = \frac{\pi(S_t, A_t)}{b(S_t, A_t)} \nu(\psi, S_t, A_t) b(S_t, A_t) = \nu(\psi, S_t, A_t) \pi(S_t, A_t).$$

This removes the importance sampling ratios from the eligibility trace. The resulting ABTD(ζ) algorithm can be written as the standard Off-policy TD(λ) algorithm, for a particular setting of λ. The Off-policy TD(λ) algorithm, with this λ, is called ABTD(ζ), with updates

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} \rho_t\big(R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t\big) \\
\mathbf{z}_t &\leftarrow \gamma_t \nu_{t-1} \pi_{t-1} \mathbf{z}_{t-1} + \mathbf{x}_t \quad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \rho_t \delta_t \mathbf{z}_t.
\end{aligned}$$

Finally, we can adapt Retrace(λ) and Tree Backup(λ) for policy evaluation. Mahmood, Yu, and Sutton (2017) showed that Retrace(λ) can be specified with a particular setting of ν_t (in their Equation 36). We can similarly obtain Retrace(λ) for prediction by setting

$$\nu_{t-1} = \zeta \min\left(\frac{1}{\pi_{t-1}}, \frac{1}{b_{t-1}}\right),$$

or more generally:

$$\nu_{t-1} = \zeta \min\left(\frac{\bar{c}}{\pi_{t-1}}, \frac{1}{b_{t-1}}\right),$$

where $\bar{c}$ is a constant, which we will discuss in more detail shortly. For Tree Backup(λ), the setting for ν_t is any constant value in $[0, 1]$ (see Algorithm 2 of Precup, Sutton & Singh, 2000).

So far, we derived ABTD(ζ) for prediction by defining λ_t in the eligibility trace update of Off-policy TD(λ). We then used two special settings of ν

to recover V-trace(λ) and Tree Backup(λ) algorithms. Now, we specify Tree Backup(λ), and V-trace(λ) updates again, but this time in terms of a special setting of λ_t in the Off-policy TD(λ) update.

Prediction variant of Tree Backup(λ) is Off-policy TD(λ) with $\lambda_t = b_{t-1}\lambda$, for some tuneable constant $\lambda \in [0, 1]$. Replacing λ_t with $b_{t-1}\lambda$ in the eligibility trace update in (6.2) simplifies as follows:

$$\begin{aligned}\mathbf{z}_t &\leftarrow \gamma_t \frac{\pi_{t-1}}{b_{t-1}} b_{t-1}\lambda \mathbf{z}_{t-1} + \mathbf{x}_t, \\ &= \gamma_t \pi_{t-1} \lambda \mathbf{z}_{t-1} + \mathbf{x}_t. \end{aligned} \tag{6.4}$$

A simplified variant of the V-trace(λ) algorithm (Espeholt et al., 2018) can be derived with a similar substitution:

$$\lambda_t = \min\left(\frac{\bar{c}}{\pi_{t-1}}, \frac{1}{b_{t-1}}\right) \lambda b_{t-1},$$

where $\bar{c} \in \mathbb{R}^+$ and $\lambda \in [0, 1]$ are both tuneable constants. The update rule for the eligibility trace of V-trace(λ) with this special setting of λ_t at each time step becomes:

$$\begin{aligned}\mathbf{z}_t &= \gamma_t \min\left(\frac{\bar{c}}{\pi_{t-1}}, \frac{1}{b_{t-1}}\right) \lambda b_{t-1} \frac{\pi_{t-1}}{b_{t-1}} \mathbf{z}_{t-1} + \mathbf{x}_t \\ &= \gamma_t \min\left(\frac{\bar{c}}{\pi_{t-1}}, \frac{1}{b_{t-1}}\right) \lambda \pi_{t-1} \mathbf{z}_{t-1} + \mathbf{x}_t \\ &= \gamma_t \min\left(\frac{\bar{c}\pi_{t-1}}{\pi_{t-1}}, \frac{\pi_{t-1}}{b_{t-1}}\right) \lambda \mathbf{z}_{t-1} + \mathbf{x}_t \\ &= \gamma_t \min\left(\bar{c}, \rho_{t-1}\right) \lambda \mathbf{z}_{t-1} + \mathbf{x}_t. \end{aligned} \tag{6.5}$$

The parameter $\bar{c}$ is used to clip importance sampling ratios in the trace. Note that it is not possible to recover the full V-trace(λ) algorithm in this way. The more general V-trace(λ) algorithm uses an additional parameter, $\bar{\rho} \in \mathbb{R}^+$ that clips the ρ_t in the update to $\mathbf{w}_{t+1}$: $\min(\bar{\rho}, \rho_t)\delta_t \mathbf{z}_t$. When $\bar{\rho}$ is set to the largest possible importance sampling ratio, it does not affect ρ_t in the update to $\mathbf{w}_t$ and so we obtain the equivalence above. For smaller $\bar{\rho}$, however, V-trace(λ) is no longer simply an instance of Off-policy TD(λ). In our experiments, we investigate this simplified variant of V-trace(λ) that does not clip ρ_t and set $\bar{c} = 1$ as done in the original Retrace algorithm.

Finally, as mentioned before, ABTD(ζ) for $\zeta \in [0,1]$ uses $\lambda_t = \nu_{t-1} b_{t-1}$ in the Off-policy TD(λ) update which results in the following eligibility trace update:

$$\begin{aligned}\mathbf{z}_t &= \gamma_t \frac{\pi_{t-1}}{b_{t-1}} \nu_{t-1} b_{t-1} \mathbf{z}_{t-1} + \mathbf{x}_t \\ &= \gamma_t \nu_{t-1} \pi_{t-1} \mathbf{z}_{t-1} + \mathbf{x}_t, \end{aligned} \tag{6.6}$$

The convergence properties of all three methods are similar to Off-policy TD(λ). They are not guaranteed to converge under off-policy sampling with weighting μ_b and function approximation. With the addition of gradient corrections similar to GTD(λ), all three algorithms are convergent. For explicit theoretical results, see Mahmood, Yu, and Sutton (2017) for ABQ(ζ) with gradient correction and Touati et al. (2018) for convergent versions of Retrace(λ) and Tree Backup(λ).

Trying to simply derive a control variant of these algorithms without taking into account which terms need to be corrected in prediction versus in control learning, might result in deriving an incorrect version of the algorithm that we briefly discuss below.

An alternative but incorrect extension of ABQ(ζ) to ABTD(ζ) The ABQ(ζ) algorithm specifies λ to ensure that $\rho_t \lambda_t$ is well-behaved, whereas we specified λ so that $\rho_t \lambda_{t+1}$ is well-behaved. This difference arises from the fact that for action-values, the immediate reward and next state are not re-weighted with ρ_t. Consequently, the λ-return of a policy from a given state and action is:

$$R_{t+1} + \gamma_{t+1} \left[(1 - \lambda_{t+1}) \hat{v}(S_{t+1}) + \rho_{t+1} \lambda_{t+1} G^{\lambda}_{t+1} \right].$$

To mitigate variance in ABQ(ζ) when learning action-values, therefore, λ_{t+1} should be set to ensure that $\rho_{t+1} \lambda_{t+1}$ is well-behaved. For ABTD(ζ), however, λ_{t+1} should be set to mitigate variance from ρ_t rather than from ρ_{t+1}.

To see why more explicitly, the central idea of these algorithms is to avoid importance sampling altogether: this choice ensures that the eligibility trace

does not include importance sampling ratios. The eligibility trace $\mathbf{z}_t^a$ in TD when learning action-values is:

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

for state-action features $\mathbf{x}_t^a$. For $\rho_t \lambda_t = \nu_t \pi_t$, this trace reduces to $\mathbf{z}_t^a = \nu_t \pi_t \gamma_t \mathbf{z}_{t-1}^a + \mathbf{x}_t^a$ (Equation 18, Mahmood et al., 2017). For ABTD(ζ), one could in fact also choose to set λ_t so that $\rho_t \lambda_t = \nu_t \pi_t$ instead of $\rho_t \lambda_{t+1} = \nu_t \pi_t$. However, this would result in eligibility traces that still contain importance sampling ratios. The eligibility trace in TD when learning state-values is:

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

Setting $\rho_t \lambda_t = \nu_t \pi_t$ would result in the update $\mathbf{z}_t = \rho_{t-1} \nu_t \frac{\pi_t}{\rho_t} \gamma_t \mathbf{z}_{t-1} + \mathbf{x}_t$, which does not remove important sampling ratios from the eligibility trace. Rather, the corresponding update for policy evaluation requires $\rho_{t-1} \lambda_t = \nu_{t-1} \pi_{t-1}$, giving the ABTD($\zeta$) as specified above.

6.3 TDRC with General λ

One of the algorithms that we will use in the empirical studies is TDRC(λ). In Chapter 4 we introduced the TDRC algorithm for the full bootstrapping case.

Remember from Chapter 4 that the update rule for TDRC's secondary weight vector is:

$$\mathbf{v}_{t+1} \leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}} \Big[\delta_t \mathbf{x}_t - (\mathbf{v}_t^\top \mathbf{x}_t) \mathbf{x}_t \Big] - \alpha_{\mathbf{v}} \mathbf{v}_t. \tag{6.7}$$

The intuition for this update is that we need the correction term in the TD update, but we would like the correction term to be small in magnitude, and we control its magnitude by regularizing the update for the second weight vector and keeping it small. This is what the regularization term in (6.7) does. Given that we want the same effect when general λ is used, the update for the secondary weight vector of the TDRC(λ) algorithm is:

$$\mathbf{v}_{t+1} \leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}} \Big[\delta_t \mathbf{z}_t - (\mathbf{v}_t^\top \mathbf{x}_t) \mathbf{x}_t \Big] - \alpha_{\mathbf{v}} \mathbf{v}_t,$$

which similar to the full bootstrapping case, subtracts a multiple of the secondary weight vector from the update to keep the magnitude of $\mathbf{v}$ small. The update rule for the primary weight vector of TDRC(λ) is the same as GTD(λ). The TDRC(λ) algorithm is fully specified by the following update rules:

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_t(\gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}_t) \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{v}_{t+1} &\leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}} \Big[\delta_t \mathbf{z}_t - (\mathbf{v}_t^\top \mathbf{x}_t)\mathbf{x}_t \Big] - \alpha_{\mathbf{v}} \mathbf{v}_t \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t - \alpha \gamma_{t+1}(1 - \lambda_{t+1})(\mathbf{v}_t^\top \mathbf{z}_t)\mathbf{x}_{t+1}.
\end{aligned}$$

With the introduction of TDRC(λ), we have now introduced all the algorithms that will be used in the empirical studies.

6.4 The Collision Task

The Collision task is an idealized off-policy prediction-learning task. A vehicle moves along an eight-state track towards an obstacle with which it will collide if it keeps moving forward. In this episodic task, each episode begins with the vehicle in one of the first four states (selected at random with equal probability). In these four states, `forward` is the only possible action whereas, in the last four states, two actions are possible: `forward` and `turnaway` (see Figure 6.1). The `forward` action always moves the vehicle one state further along the track; if it is taken in the last state, then a collision is said to occur, the reward is 1, and the episode ends. The `turnaway` action causes the vehicle to "turn away" from the wall, which also ends the episode, except with a reward of zero. The reward is also zero on all earlier, non-terminating transitions. In an episodic task like this the return is accumulated only up to the end of the episode. After termination, the next state is the first state of the next episode, selected randomly from the first four as specified above.

The target policy on this task is to always take the forward action, meaning that $\pi(\texttt{forward}|s) = 1, \forall s \in \mathcal{S}$, whereas the behavior policy is to take the two actions (where available) with equal probability, which means that

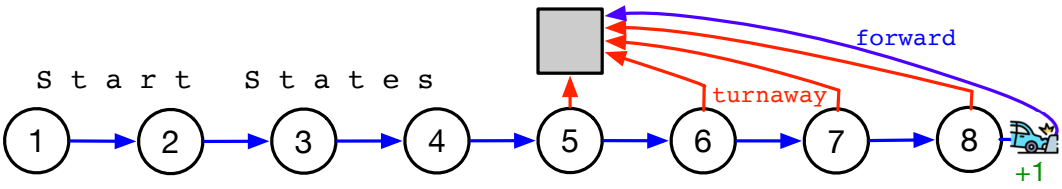


Figure 6.1: The Collision task. Episodes start in one of the first four states and end when the `forward` action is taken from the eighth state, causing a crash and a reward of 1, or when the `turnaway` action is taken in one of the last four states.

$b(\texttt{forward}|s) = b(\texttt{turnaway}|s) = 0.5, \forall s \in \{5, 6, 7, 8\}$. The problem is discounted with a discount rate of $\gamma = 0.9$. As always, we are seeking to learn the value function for the target policy, which in this case is $v_\pi(s) = \gamma^{8-s}$. This function is shown as a dotted black curve in Figure 6.2. The thin red lines show approximate value functions $\hat{v} \approx v_\pi$, using various feature representations, as we discuss shortly below.

This idealized task is roughly analogous to and involves some similar issues as real-world autonomous driving problems, such as exiting a parallel parking spot without hitting the car in front of you, or learning how close you can get to other cars without risking collisions. In particular, if these problems can be treated as off-policy learning problems, then solutions can potentially be learned with fewer collisions. In this work, we are testing the efficiency of various off-policy prediction-learning algorithms at maximizing how much they learn from the same number of collisions.

Similar problems have been studied using mobile robots. For example, White (2015) used off-policy learning algorithms running on an iRobot Create to predict collisions as signaled by activation of the robot's front bumper sensor. Rafiee et al. (2019) used a Kobuki robot to predict predictions as well. Modayil and Sutton (2014) trained a custom robot to predict motor stalls and turn off the motor when a stall was predicted.

We artificially introduce function approximation into the Collision task. Although a tabular approach is entirely feasible on this small problem, it would not be on the large problems of interest. In real applications, the agent

would have sensor readings, which will go through an artificial neural network to create feature representations. We simulate such representations in the Collision task by randomly assigning to each of the eight states a binary feature vector $\mathbf{x}(s) \in \{0,1\}^d, \forall s \in \{1..8\}$. We chose $d = 6$, so that was not possible

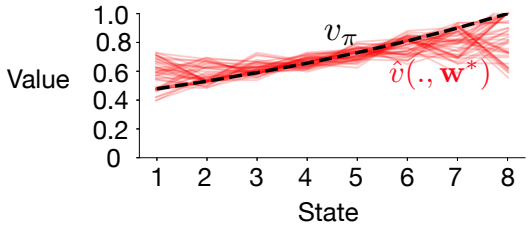


Figure 6.2: The ideal value function, v_π, and the best approximate value functions, $\hat{v}$, for 50 different feature representations.

for all eight of the feature vectors (one per state) to be linearly independent. In particular, we chose all eight feature vectors to have exactly three 1s and three 0s, with the location of the 1s for each state being chosen randomly.

Because the feature vectors are linearly dependent, it is not possible in general for a linear approximation, $\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$, to equal to $v_\pi(s)$ at all eight states of the Collision task. This, in fact, is the sole reason the red approximate value functions in Figure 6.2 do not exactly match v_π. Given a feature representation $\mathbf{x} : \mathcal{S} \to \mathbb{R}^d$, a linear approximate value function is completely determined by its weight vector $\mathbf{w} \in \mathbb{R}^d$. The quality of that approximation is assessed by its squared error at each state, weighted by how often each state occurs:

$$\overline{\text{VE}}(\mathbf{w}) = \sum_{s \in \mathcal{S}} \mu_b(s) \big[\hat{v}(s, \mathbf{w}) - v_\pi(s)\big]^2, \tag{6.8}$$

where $\mu_b(s)$ is the state distribution, the fraction of time steps in which $S_t = s$, under the behavior policy (here μ_b was approximated from visitation counts from one million sample time steps). The value functions shown by red lines in Figure 6.2 are for $\mathbf{w}^*$, the weight vector that minimizes $\overline{\text{VE}}(\mathbf{w})$, with each line corresponding to a different randomly selected feature representation as described earlier. For these value functions, $\overline{\text{VE}}(\mathbf{w}^*) \approx 0.05$. The code for

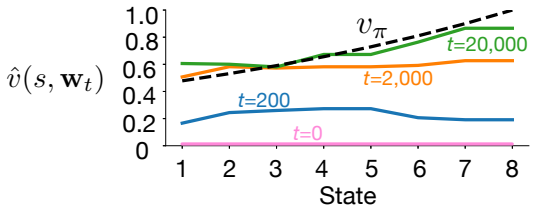


Figure 6.3: An example of the approximate value function, $\hat{v}$, being learned over time.

the Collision task and the experiments are provided at the following link: `https://github.com/sinaghiassian/OffpolicyAlgorithms`.

6.5 Experiment

The Collision task, in conjunction with its behavior policy, was used to generate 20,000 time steps, comprising one *run*, and then this was repeated for a total of 50 independent runs. Each run also used a different feature representation randomly generated as described in the previous section. The eleven learning algorithms were then applied to the 50 runs, each with a range of parameter values; each combination of algorithm and parameter settings is termed an *algorithm instance*. A list of all parameter settings used can be found in Table 6.1. They included 12 values of λ, 19 values of α, 15 values of η (for the Gradient-TD family), six values of β (for ETD(λ, β)), and 12 values of ζ (for ABTD(ζ)), for approximately 20,000 algorithm instances in total. In each run, the weight vector was initialized to $\mathbf{w}_0 = \mathbf{0}$ and then updated at each step by the algorithm instance to produce a sequence of $\mathbf{w}_t$. At each step we also computed and recorded $\overline{\text{VE}}(\mathbf{w}_t)$. With a successful learning procedure, we expect the value function to evolve over time as in Figure 6.3. The approximate value function starts at $\hat{v}(s, \mathbf{0}) = 0$, as shown by the pink line, then moves toward positive values, as shown by the blue and orange lines. Finally, the learned value function slants and comes to closely approximate the true value function, though always with some residual error due to the limited feature representation, as shown by the green line (and also by all the red lines in Figure 6.2).

<table>
<tr><th colspan="2">Algorithms</th><th>η or β</th><th>λ or ζ</th><th>α</th></tr>
<tr><td colspan="2">Off-policy TD(λ)</td><td>—</td><td rowspan="9">0,
0.1, 0.2,
0.3, 0.5,
0.9, 1
and
1 - 2^{-x}
where
$x \in \{$
$2, 3, 4, 5, 6\}$</td><td rowspan="9">$\alpha = 2^{-x}$
where
$x \in \{$
$0, 1, 2, \cdots,$
$17, 18\}$</td></tr>
<tr><td rowspan="5">Gradient-TD
Algorithms</td><td>GTD(λ)</td><td rowspan="4">2^x
where $x \in$
$\{-6, -5, \cdots,$
$7, 8\}$</td></tr>
<tr><td>GTD2(λ)</td></tr>
<tr><td>HTD(λ)</td></tr>
<tr><td>Proximal GTD2(λ)</td></tr>
<tr><td>TDRC(λ)</td><td>—</td></tr>
<tr><td rowspan="2">Emphatic-TD
Algorithms</td><td>Emphatic
TD(λ)</td><td>—</td></tr>
<tr><td>Emphatic
TD(λ, β)</td><td>$\beta \in$
$\{0.0, 0.2,$
$0.4, 0.6,$
$0.8, 1.0\}$</td></tr>
<tr><td rowspan="3">Variable-λ
Algorithms</td><td>Tree Backup(λ)</td><td rowspan="3">—</td></tr>
<tr><td>V-trace(λ)</td></tr>
<tr><td>ABTD(ζ)</td></tr>
</table>

Table 6.1: List of all parameters that we used in the experiment. For algorithms such as GTD(λ) that had more than one parameter, we tried all the possible combinations of all parameters.

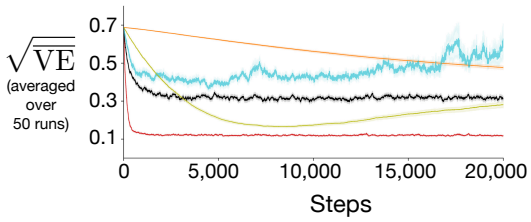


Figure 6.4: Learning curves illustrating the range of things that can happen during a run. The average error over the 20,000 steps is a good combined measure of learning rate and asymptotic error.

Figure 6.4 shows learning curves illustrating the range of things that happened in the experiment. Normally, we expect $\overline{\text{VE}}$ to decrease over the course of the experiment, starting at $\overline{\text{VE}}(\mathbf{0}) \approx 0.7$ and falling to some minimum value, as in the red and black lines in Figure 6.4 (these and all other data are averaged over the fifty runs). If the primary step-size parameter, α, is small, then the learning may be slow and incomplete by the end of the runs, as in the orange line. A larger step-size parameter may be faster, but, if it is too large, then divergence can occur, as in the blue line. For one algorithm, Proximal GTD2(λ), we found that the error dipped low and then leveled off at a higher level, as in the olive line.

6.6 Main Results: A Partial Order over Algorithms

As an overall measure of the performance of an algorithm instance, we take its learning curve over 50 runs, as in Figure 6.4, and then average it across the 20,000 steps. In this way, we reduce all the data for an algorithm instance to a single number that summarizes performance. These numbers appear as points in our main results figure, Figure 6.5. Each panel of the figure is devoted to a single algorithm.

For example, performance numbers for instances of Off-policy TD(λ) are shown as points in the left panel of the second row of Figure 6.5. This algorithm has two parameters, the step-size parameter, α, and the bootstrapping parameter, λ. The points are plotted as a function of α, and points with the same

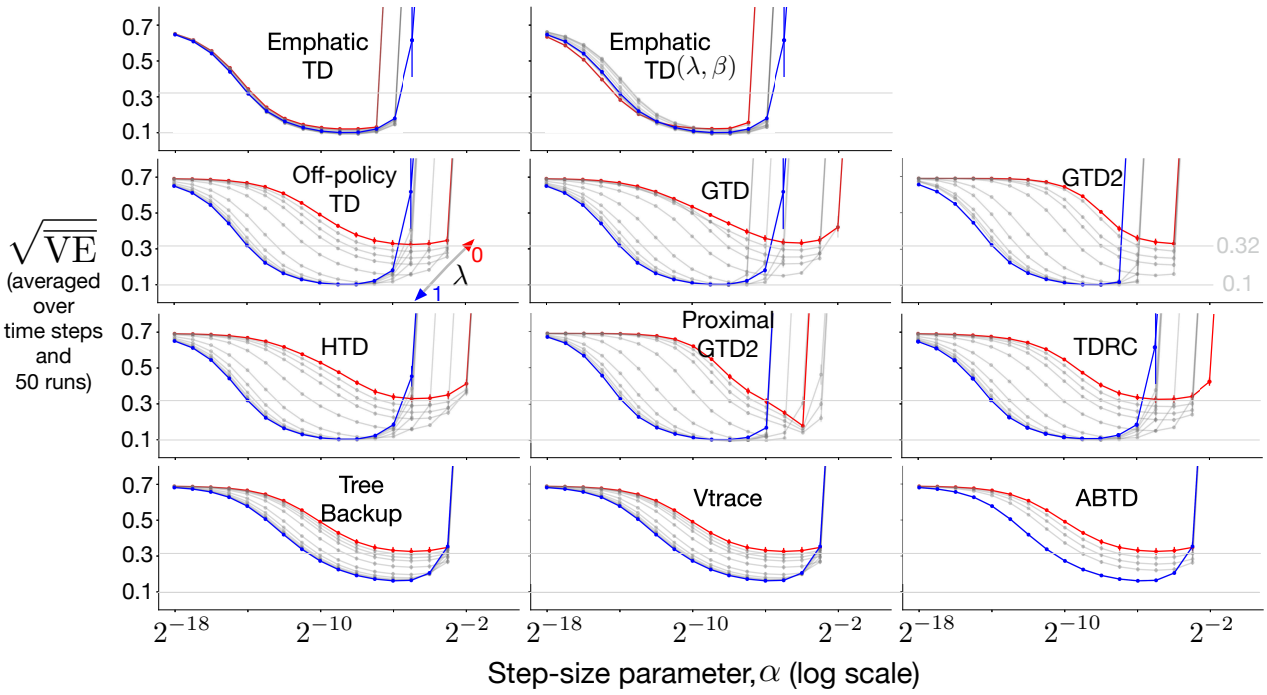


Figure 6.5: Main results: Performance of all algorithms on the Collision task as a function of their parameters α and λ. The top tier algorithms (top row) attained a low error ($\approx$0.1) at all λ values. The middle tier of six algorithms attained a low error for $\lambda = 1$, but not for $\lambda = 0$. And the bottom-tier of three algorithms were unable to reach an error of $\approx$0.1 at any λ value.

λ value are connected by lines. The blue line shows the performances of the instances of Off-policy TD(λ) with $\lambda = 1$, the red line shows the performances with $\lambda = 0$, and the gray lines show the performances with intermediate λs. Note that all the lines are U-shaped functions of α, as is to be expected; at small α learning is too slow to make much progress, and at large α there is overshoot and divergence, as in the blue line in Figure 6.4. For each point, the standard error over the 50 runs is also given as an error bar, though these are too small to be seen in all except the rightmost points of each line where the step size was highest and divergence was common. Except for these rightmost points, almost all visible differences are statistically significant.

First focus on the blue line (of the left panel on the second row of Figure 6.5), representing the performances of Off-policy TD(λ) with $\lambda = 1$. There is a wide sweet spot, that is, there are many intermediate values of α at which

good performance (low average error) is achieved. Note that the step-size parameter α is varied over a wide range, with logarithmic steps. The minimal error level of about 0.1 was achieved over four or five powers of two for α. This is the primary measure of good performance that we look for in these data: low error over a wide range of parameter values.

Now contrast the blue line with the red and gray lines (for Off-policy TD(λ) in the left panel of the second row of Figure 6.5). Recall that the blue line is for $\lambda = 1$, the red line is for $\lambda = 0$, and the gray lines are for intermediate values of λ. First note that the red line shows generally worse performance; the error level at $\lambda = 0$ was higher, and its range of good α values was slightly smaller (on a logarithmic scale). The intermediate values of λ all had performances that were between the two extremes. Second, the sweet spot (the best α value) consistently shifted right, toward higher α, as λ was decreased from 1 toward 0.

Now, armed with a thorough understanding of the Off-policy TD(λ) panel, consider the other panels of Figure 6.5. Overall, there are a lot of similarities between the algorithms and how their performances varied with α and λ. For all algorithms, error was lower for $\lambda = 1$ (the blue line) than for $\lambda = 0$ (the red line). Bootstrapping apparently confers no advantage in the Collision task for any algorithm.

The most obvious difference between algorithms is that the performance of the two Emphatic-TD algorithms varied relatively little as a function of λ; their blue and red lines are almost on top of one another, whereas those of all the other algorithms are qualitatively different. The emphatic algorithms generally performed as well as or better than the other algorithms. At $\lambda = 1$, the emphatic algorithms reached the minimal error level of all algorithms ($\approx$0.1), and their ranges of good α values was as wide as that of the other algorithms. While at $\lambda = 0$, the best errors of the emphatic algorithms were qualitatively better than those of the other algorithms. The minimal $\lambda = 0$ error level of the emphatic algorithms was about 0.15, as compared to approximately 0.32 (shown as a second thin gray line) for all the other algorithms (except Proximal GTD2, a special case that we consider later). Moreover, for the

emphatic algorithms the sweet spot for α shifted little as λ varied. The shift was markedly less than for the six algorithms in the middle two rows of Figure 6.5. The lack of an interaction between the two parameter values is another potential advantage of the emphatic algorithms.

The lowest error level for eight of the algorithms was $\approx$0.1 (shown as a thin gray line), and for the other three algorithms the best error was higher, $\approx$0.16. The differences between the eight and the three were highly statistically significant, whereas the differences within the two groups were negligible. The three algorithms that performed worse than the others were Tree Backup(λ), V-trace(λ), and ABTD(ζ)—shown in the bottom row of Figure 6.5. The difference was only for large λs; at $\lambda = 0$ these three algorithms reached the same error level ($\approx$0.32) as the other non-emphatic algorithms. The three worse algorithms' range of good α values was also slightly smaller than the other algorithms (with the partial exception, again, of Proximal GTD2(λ)). A mild strength of the three is that the best α value shifted less as a function of λ than for the other six non-emphatic algorithms. Generally, the performances of these three algorithms in Figure 6.5 look very similar as a function of parameters. An interesting difference is that for ABTD(ζ), we only see three gray curves, whereas for the other two algorithms we see seven. For ABTD(ζ) there is no λ parameter, but the parameter ζ plays the same role. In our experiment, ABTD(ζ) performed identically for all ζ values greater than 0.5; four gray lines with different ζ values are hidden behind ABTD's blue curve.

In summary, our main result is that on the Collision task the performances of the eleven algorithms fell into three groups, or tiers. In the top tier are the two Emphatic-TD algorithms, which performed well and almost identically at all values of λ and significantly better than the other algorithms at low λ. Although this difference did not affect best performance here (where $\lambda = 1$ is best), the ability to perform well with bootstrapping is expected to be important on other tasks. In the middle tier are Off-policy TD(λ) and all the Gradient-TD algorithms including HTD(λ), all of which performed well at $\lambda = 1$ but less well at $\lambda = 0$. Finally, in the bottom tier are Tree Backup(λ), V-trace(λ), and ABTD(λ), which performed very similarly and not as well as

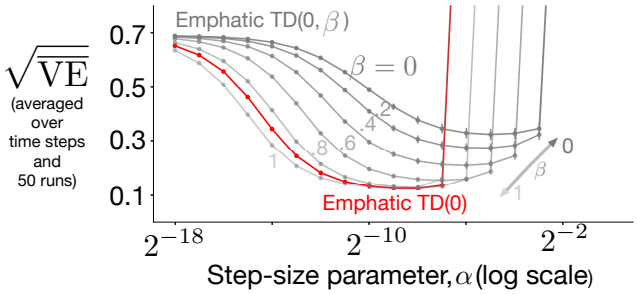


Figure 6.6: Detail on the performance of Emphatic TD(λ, β) at $\lambda = 0$. Note that Emphatic TD(λ) is equivalent to Emphatic TD(λ, γ), and here $\gamma = 0.9$. The flexibility provided by β does not help on the Collision task.

the other algorithms at their best parameter values. All of these differences are statistically significant, albeit specific to this one task. In Figure 6.5 the three tiers are the top row, the two middle rows, and the bottom row.

In the next two sections we take a closer look at two of the tiers to find differences within them.

6.7 Emphatic TD(λ) versus Emphatic TD(λ, β)

In this section, the effect of the β parameter of Emphatic TD(λ, β) on the algorithm's performance in the full bootstrapping case is analyzed. We focus on the full bootstrapping case ($\lambda = 0$) because this is where the largest differences were observed in the previous section. The curves shown in Figure 6.5, are for the best values of β; meaning that, for each λ, we found the combination of α and β that resulted in the minimum average error, fixed β, and plotted the sensitivity for that fixed β over the step-size parameter. Here, we show how varying β affects performance.

The error of Emphatic TD(0), and Emphatic TD(0,β) for various values of α and β are shown in Figure 6.6. We see that both algorithms performed similarly well on the Collision task, meaning that they both had a wide sensitivity curve and reached the same ($\approx$0.1) error level. Notice that, as β increased, the sensitivity curve for Emphatic TD(0,β) shifted to left and the overall error decreased. With $\beta = 0$, Emphatic TD(λ, β), reduces to TD(λ). With $\beta = 0.8$,

and $\beta = 1$, Emphatic TD(λ, β) reached the same error level as Emphatic TD(λ). With $\beta = \gamma$, Emphatic TD(λ, β) reduces to Emphatic TD(λ). This explains why the red curve is between the $\beta = 0.8$ and $\beta = 1$ curves.

The results make it clear that the superior performance of emphatic methods are almost entirely due to the basic idea of emphasis; the additional flexibility provided by β of the Emphatic TD(λ, β) was not important on the Collision problem.

6.8 Assessment of Gradient-TD Algorithms

We study how the η parameter of Gradient-TD algorithms affects performance in the case of full bootstrapping (the second step-size parameter, $\alpha_{\mathbf{v}}$, is equal to $\eta \times \alpha$). Previously, in Figure 6.5 we looked at the results with the best values of η for each λ; meaning that for each λ, first the combination of α and η that resulted in the lowest average $\overline{\text{VE}}$ was found and then sensitivity to the step-size parameter was plotted for that specific value of η. Sensitivity to step size for various values of η for $\lambda = 0$ are shown in Figure 6.7. Each panel shows the result of two Gradient-TD algorithms for various η. One main algorithm, shown with solid lines, and another additional algorithm shown with dashed lines for comparison. First focus on the upper left panel. The upper left panel shows the parameter sensitivity for GTD2(0), for four values of η, and additionally it shows GTD(0) results as dashed lines for comparison. We plotted these four values out of the eight values listed in Table 6.1 because they are a good representative of all the value of η. The color for each value of η is consistent within and across the four panels, meaning that for example, $\eta = 256$ is shown in green in all panels, either as dashed or solid lines. For all parameter combinations, GTD errors were lower than (or similar to) GTD2 errors. With two smaller values of η (1 and 0.0625) GTD had a wider and lower sensitivity curve than GTD2, which means GTD was easier to use than GTD2.

Let us now move on to the upper right panel of Figure 6.7. Proximal GTD2 had the most distinctive behavior among Gradient-TD algorithms. As

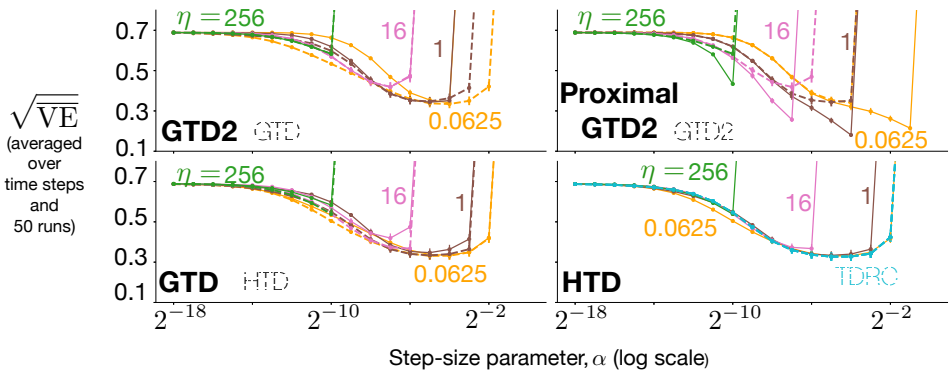


Figure 6.7: Detail on the performance of Gradient-TD algorithms at $\lambda = 0$. Each algorithm has a second step-size parameter, scaled by η. A second algorithm's performance is also shown in each panel, with dashed lines, for comparison.

previously observed in Figure 6.4, it is the only algorithm that in some cases had a "bounce"; its error dipped down at first and then moved back up. With $\lambda = 0$, it sometimes converged to an error that was lower than all other Gradient-TD algorithms. Proximal GTD2 was more sensitive to the choice of α than other Gradient-TD algorithms except GTD2. Proximal GTD2 had a lower error and a wider sensitivity curve than GTD2. To see this, compare the dotted and solid lines in the upper right panel of Figure 6.7.

Moving on to the lower left panel, we see that GTD and HTD performed similarly. Sensitivity curves were similarly wide but HTD reached a lower error in some cases. We see this by comparing the dotted and solid pink curves in the lower left panel.

The fourth panel shows sensitivity to the step-size parameter for HTD and TDRC. Notice that TDRC has one sensitivity curve, shown in dashed blue. This is because η is set to one (also its regularization parameter was set to one) as proposed in the original paper. HTD's widest curve was with $\eta = 0.0625$ which was as wide as TDRC's curve.

On one hand, among the Gradient-TD algorithms, TDRC was the easiest to use. On the other hand, in the case of full bootstrapping, Proximal GTD2 reached the lowest error level among all Gradient-TD algorithms. It remains to be seen how these algorithms compare on other problems.

6.9 Conclusions

We conducted an empirical study of off-policy prediction learning algorithms on the Collision task. We learned that Emphatic TD(λ) is one of the best algorithms for solving this task, as it learns the fastest, converges to the lowest error, and is not more sensitive to parameters than other algorithms. Using V-trace, Tree Backup, and ABTD to solve the Collision task did not provide any benefit over using other algorithms. These three algorithms did not learn faster, and they converged to a higher asymptotic error than other algorithms. Within the Gradient-TD family, TDRC(λ) seems to be the easiest to use algorithm as it provides a standard way for choosing the second step-size parameter, and it performs similarly to other Gradient-TD algorithms.

The present study is based on a single task, and this limits the conclusions that can be fairly drawn from it. For example, we have found that Emphatic-TD methods perform well over a wider range of parameters than Gradient-TD methods on the Collision task, but it is entirely possible that the reverse would be true on a different task. Many more tasks must be explored before it is possible for a consistent pattern to emerge that favors one class of algorithm over another.

On the other hand, a pattern over empirical results must begin somewhere. We stress the need for extensive empirical results even for a single task. Ours is the first systematic study of off-policy learning to describe the effects of all algorithm parameters individually (rather than, for example, taking the best performing parameters or fixing one parameter and studying another).

Chapter 7

An Empirical Comparison in the Four Rooms Environment[1]

One of the main objectives of this dissertation is to empirically study off-policy prediction learning algorithms. So far, we empirically studied eleven prominent off-policy prediction learning algorithms on the Collision task. This chapter presents another important contribution of this dissertation: an empirical study of off-policy prediction learning algorithms with a focus on the challenge of high variance in off-policy learning.

In this chapter, we empirically compare off-policy prediction learning algorithms on two small tasks: the Rooms task, and the High Variance Rooms task. Both tasks are based on the original Four Rooms environment proposed by Sutton, Precup, and Singh (1999). The tasks are designed such that learning fast in them is challenging. In the Rooms task, the product of importance sampling ratios can be as large as 2^{14} and in the High Variance Rooms task, the product of the ratios can become as large as $2^{14} \times 25$. To control the high variance caused by the product of the importance sampling ratios, the step-size parameter should be set small, which in turn slows down learning. The experiments conducted in this chapter build on the ones from the previous chapter. We consider the same set of algorithms as in the previous chapter and employ the same experimental methodology. Based on the data, we conclude that the performance of most algorithms is highly affected by the variance induced by

[1]The contents of this chapter are based on a paper co-authored by this author (Ghiassian & Sutton, 2021b).

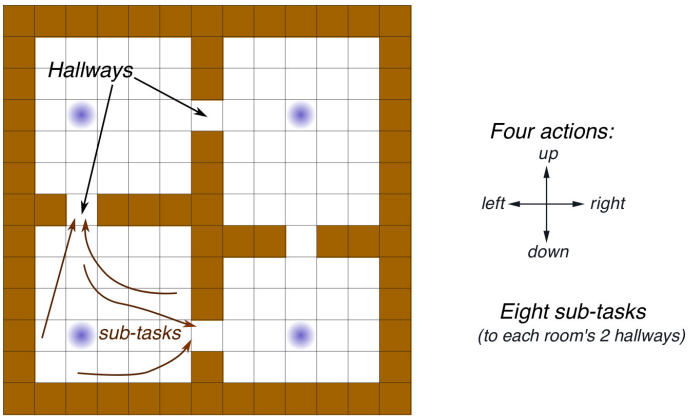


Figure 7.1: The Four Rooms environment. Four actions are possible in each state. Two hallway states are shown with arrows. Four sub-tasks are also schematically shown. The four shaded states are the ones where the Rooms and the High Variance Rooms tasks have different policies.

the importance sampling ratios. We found that the algorithms that adapt the λ parameter of Off-policy TD(λ) (Tree Backup(λ), V-trace(λ), and ABTD(ζ)) are not affected by the high variance as much as the other algorithms. We observed that Emphatic TD(λ) tends to have lower asymptotic error than other algorithms, similar to what we observed in the Collision task, but it tends to learn more slowly in problems where the product of importance sampling ratios is high.

7.1 Rooms Task

The Rooms task uses a variant of the Four Rooms environment MDP (Sutton, Precup, & Singh, 1999): a gridworld with 104 states, roughly partitioned into four contiguous areas called rooms (Figure 7.1). These rooms are connected through four hallway states. Four deterministic actions are available in each state: `left`, `right`, `up`, and `down`. Taking each action results in moving in the corresponding direction except for cells neighboring a wall in which the agent will not move if it takes the action toward the wall. The task is continuing.

The Rooms task consists of eight sub-tasks. Solving a sub-task corresponds to learning the value function for the target policy corresponding to the sub-

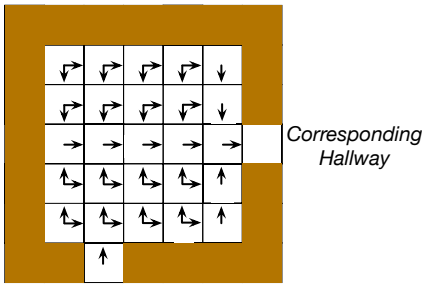


Figure 7.2: Target policy of the upper left room leading to one of the hallways.

task. Under the corresponding target policy, the agent follows a shortest path to one of the room's hallways, which we refer to as the corresponding hallway. If two actions are optimal in a state one of the two is taken randomly with equal probability. For a sample target policy defined in the upper left room, see Figure 7.2. The termination function returns 0.9 while the agent is in the room. Once the agent reaches the corresponding hallway, the termination function returns zero, without affecting the actual trajectory of the behavior policy. The termination function remains zero for all states that are not part of the sub-task. When the agent reaches the corresponding hallway, it receives a reward of +1. The rewards for all other transitions are zero. Each of these sub-tasks can also be explained using the general value function (GVF) language (Sutton et al. 2011). In the same room, another target policy is defined under which the agent follows the shortest path to the other hallway state. This means that there are exactly two sub-tasks defined for each state, including the hallways.

The Rooms task is designed to be a high variance problem. By a high variance problem, we mean the product of importance sampling ratios can vary between small and large values during learning and can cause large changes in the learned weight vector that might make learning unstable. Under the target policy, the agent follows a shortest path to a hallway, and the behavior policy is equiprobable random. If the agent is in the top left state of the upper right room, chooses the `right` action twice, and then chooses the `down` action six times, the product of the importance sampling ratios will become 2^{14} because the importance sampling ratio is 2 at the first two time steps and it is $\frac{1}{1/4} = 4$

for a total of 6 steps.

The agent learns about two sub-tasks at each time step. For Emphatic TD, this will be automatically enforced with the interest function set to 0 for all states that are not part of the active sub-task. Other algorithms do not natively have interest so we enforce this manually by making sure that at each time step, the agent knows what sub-tasks are active and only updates weight vectors corresponding to those sub-tasks.

We used linear function approximation to solve the task. To represent states, (x, y) coordinates were tile coded. The x coordinate ranged from 0 to 10, where 0 was assigned to the far left cell. The y coordinate also ranged from 0 to 10, where 0 was the bottom cell. We used four tilings, each of which was two by two tiles. In fact, the features used to solve the task can be produced using any system, for example, a neural network. Our focus, in this experiment, is on learning the value function linearly using known features, the task that is typically carried out by the last layer of the neural network.

To assess the quality of the value function found by an algorithm, we used the mean squared value error:

$$\overline{\text{VE}}(\mathbf{w}) = \frac{\sum_{s\in\mathcal{S}} \mu_b(s) i(s) \left[\hat{v}(s, \mathbf{w}) - v_\pi(s)\right]^2}{\sum_{s\in\mathcal{S}} \mu_b(s) i(s)},$$

where $i(s)$, the *interest function*, $i : \mathcal{S} \to \{0, 1\}$ defines a weighting over states and $\mu_b(s)$ is an approximation of the stationary distribution under the behavior policy which was calculated by having the agent start at the bottom left corner and following the behavior policy for a hundred million time steps and computing the fraction of time the agent spent in each state. The true value function was calculated by following each of the target policies from each state to their corresponding hallway once. The interest function is one for all states where the target policy is defined. Setting $i(s)$ in the error computation ensures that prediction errors from states outside of a room do not contribute to the error computed for each sub-task. We computed the square root of $\overline{\text{VE}}$ for each policy separately and then simply averaged the errors of the eight approximate value functions to compute an overall measure of error, which we denote by A$\overline{\text{VE}}$: A$\overline{\text{VE}}(\mathbf{w}) \stackrel{\text{def}}{=} \frac{1}{8}\sum_{j=1}^{8} \sqrt{\overline{\text{VE}}(\mathbf{w}^j)}$.

7.2 Experimental Setup

The task and the behavior policy were used to generate 50,000 steps, comprising one *run*. This was repeated for a total of 50 runs. The 11 learning algorithms were applied to the 50 runs, each with a range of parameter values. A list of all parameters used can be found in Table 7.1. They included 12 values of λ, 19 values of α, 15 values of η (for the Gradient-TD family), six values of β (for Emphatic TD(λ, β)), and 12 values of ζ (for ABTD(ζ)), for approximately 20,000 algorithm instances in total. At the beginning of each run, the weight vector was initialized to $\mathbf{w}_0 = \mathbf{0}$ and then was updated at each step by the algorithm instance. At each time step, $\overline{\text{AVE}}$ was computed and recorded. The code for the Rooms and High Variance Rooms tasks and the experiments are provided at the following link: https://github.com/sinaghiassian/OffpolicyAlgorithms.

7.3 Main Results of the Rooms Experiment

Performance of an algorithm instance is summarized by one number: $\overline{\text{AVE}}$ averaged over runs and time steps. This number is shown for many algorithm instances in Figure 7.3. Each panel shows one algorithm's performance.

Let us first focus on Off-policy TD(λ) results shown in the first panel of the second row of Figure 7.3. This algorithm has two parameters: the step-size parameter, α, and the bootstrapping parameter, λ. The x-axis shows the value of α at logarithmic scale. Each curve within the panel shows the performance with one λ. The blue curve shows performance with $\lambda = 1$ and the red curve shows performance with $\lambda = 0$. Performance with intermediate values of λ are shown in gray. With small α, learning was too slow. With large α, divergence happened. This is why all the curves in the panel are U-shaped. For each point, the standard error over 50 runs is shown as a bar over the point. The error bars are often too small and are not visible.

The measure of good performance that we look for in the data is low error over a wide range of parameters. For Off-policy TD(λ), the bottom of the

<table>
<tr><th colspan="2">Algorithms</th><th>η or β</th><th>λ or ζ</th><th>α</th></tr>
<tr><td colspan="2">Off-policy TD(λ)</td><td>—</td><td rowspan="9">0,
0.1, 0.2,
0.3, 0.5,
0.9, 1
and
1 - 2^{-x}
where
$x \in \{$
$2, 3, 4, 5, 6\}$</td><td rowspan="9">$\alpha = 2^{-x}$
where
$x \in \{$
$0, 1, 2, \cdots,$
$17, 18\}$</td></tr>
<tr><td rowspan="5">Gradient-TD
Algorithms</td><td>GTD(λ)</td><td rowspan="4">2^x
where $x \in$
$\{-6, -5, \cdots,$
$7, 8\}$</td></tr>
<tr><td>GTD2(λ)</td></tr>
<tr><td>HTD(λ)</td></tr>
<tr><td>Proximal GTD2(λ)</td></tr>
<tr><td>TDRC(λ)</td><td>—</td></tr>
<tr><td rowspan="2">Emphatic-TD
Algorithms</td><td>Emphatic
TD(λ)</td><td>—</td></tr>
<tr><td>Emphatic
TD(λ, β)</td><td>$\beta \in$
$\{0.0, 0.2,$
$0.4, 0.6,$
$0.8, 1.0\}$</td></tr>
<tr><td rowspan="3">Variable-λ
Algorithms</td><td>Tree Backup(λ)</td><td rowspan="3">—</td></tr>
<tr><td>V-trace(λ)</td></tr>
<tr><td>ABTD(ζ)</td></tr>
</table>

Table 7.1: List of all parameters that we used in the experiment. For algorithms such as GTD(λ) that had more than one parameter, we tried all the possible combinations of all parameters.

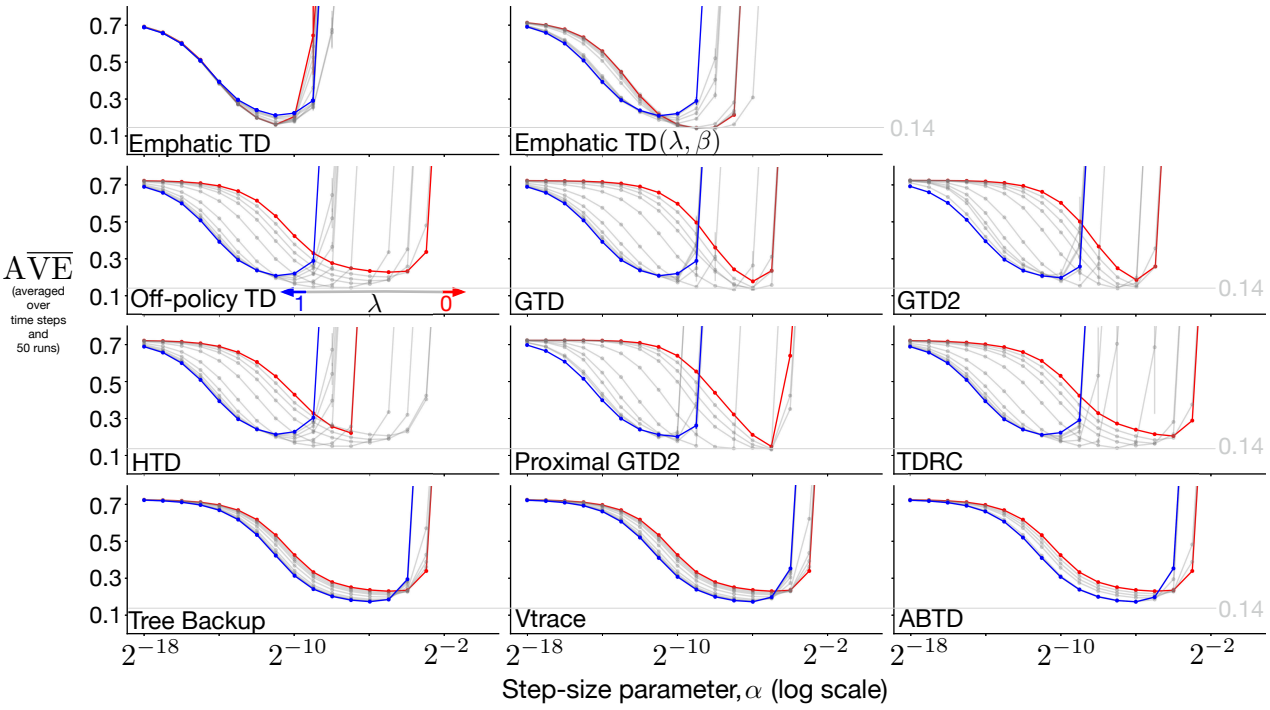


Figure 7.3: Error as a function of α and λ for all algorithms on the Rooms task. All algorithms reached the 0.14 error level except Tree Backup(λ), V-trace(λ), and ABTD(ζ). Proximal GTD2(λ) and Emphatic TD(λ) were more sensitive to α than other algorithms. Emphatic TD(λ) was less sensitive to λ than other algorithms.

U-shaped curves were at about 0.14 (shown as a thin gray line). The instances that reached this error level were in a sweet spot. This sweet spot was large for Off-policy TD(λ).

Let us now move on to studying the performance of all algorithms shown in Figure 7.3. There are lots of similarities between the algorithms. All algorithms had their best performance with intermediate values of λ, except for Tree Backup(λ), V-trace(λ), and ABTD(ζ). All algorithms except the three reached to about 0.14 error level. With all algorithms, except for Emphatic TD(λ), the sweet spot shifted to the left, as λ increased from 0 to 1. Between the five Gradient-TD algorithms shown in the two middle rows, GTD2 and Proximal GTD2 were more sensitive to α and their U-shaped curves were less smooth than some others.

One of the most distinct behaviors was observed with Emphatic TD(λ)

whose performance changed little as a function of λ. Its best performance was a bit worse than 0.14 and was achieved with $\lambda = 0$. Emphatic TD(λ) was more sensitive to α than other algorithms. Notice how its U-shaped curve is less smooth and narrower at its bottom than many others.

Tree Backup, V-trace, and ABTD behaved similarly. They did not reach the 0.14 error level, and had their best performance at λ =1. Notice how ABTD(ζ) was less sensitive to its bootstrapping parameter, ζ, and only had three gray curves, whereas V-trace(λ), and Tree Backup(λ) had more gray curves. Many of the ABTD(ζ) gray curves are hidden behind the blue curve. All these observations (not reaching the minimum error level like other algorithms, best performance with λ =1, and ABTD being more robust to the choice of the bootstrapping parameter), are consistent with what was reported from the experiments on the Collision task.

Overall, on the Rooms task, we divide the algorithms into two tiers. All algorithms except Tree Backup, V-trace, and ABTD had an error close to 0.14 and are in the first tier. Tree Backup, V-trace, and ABTD are in the second tier because regardless of how their parameters were set, they never reached the 0.14 error level. These conclusions are in some cases similar to the ones from the Collision task. When applied to the Collision task, algorithms were divided into three tiers: Emphatic-TD algorithms were in the top tier, Gradient-TD and Off-policy TD(λ) were in the middle tier, and Tree Backup, V-trace, and ABTD were in the bottom tier. Similar to the Collision task, Tree Backup, V-trace, and ABTD did not perform as well as the other algorithms when applied to the Rooms task. Unlike the Collision task, Emphatic TD's best performance was similar to Gradient-TD algorithms' best performance, but not better.

7.4 Emphatic-TD Algorithms Applied to the Rooms Task

So far, we looked at performance as a function of α and λ. We now set $\lambda = 0$, and study the effect of β on the performance of Emphatic TD(λ, β). Errors

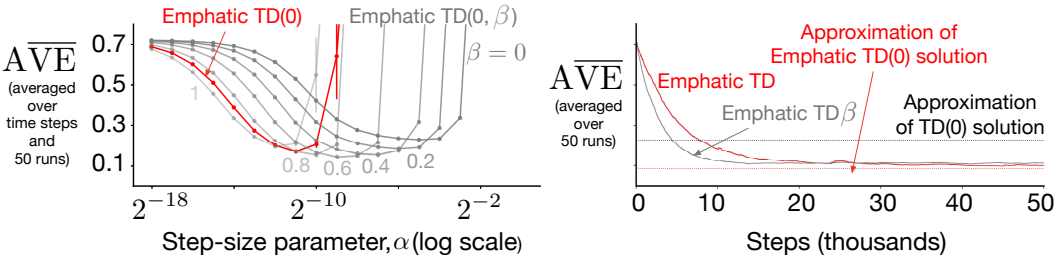


Figure 7.4: Detail on the Emphatic TD(λ, β) performance on the Rooms task at $\lambda = 0$ is shown on the left. Best learning curves for each algorithm are shown on the right. The β parameter helped Emphatic TD(λ, β) learn faster.

for various βs are plotted in the left panel of Figure 7.4. Best performance was achieved with an intermediate β and was statistically significantly lower than the error of Emphatic TD.

These results make it clear that the β parameter improves the performance of Emphatic-TD algorithms on the Rooms task. This is in contrast to the results reported from the Collision task experiment in which no improvement was observed by varying β. It seems like varying β might only be useful in cases where the problem variance is high where by problem variance we mean the variance induced by the large products of importance sampling ratios.

Two learning curves and two dotted straight lines are shown in the right panel of Figure 7.4. The learning curves correspond to algorithm instances of Emphatic TD(0) and Emphatic TD(0, β) that minimized the area under the learning curve (AUC). The dotted lines show the approximate solutions of Emphatic TD(0) and Off-policy TD(0). These solutions are found using all the data over 50,000 time steps and 50 runs, and the Least-squares algorithms discussed in Section 3.6. These solutions show the error level these algorithms would converge to if they were applied to the task with a small enough α and were run for long enough.

Emphatic TD(0) learned slower than Emphatic TD(0, β). In fact, Emphatic TD(λ) learned slower than all other algorithms when applied to the Rooms task. Learning curves for algorithm instances with the smallest AUC for of all algorithms for general λ are shown in Figure 7.5. However, if Em-

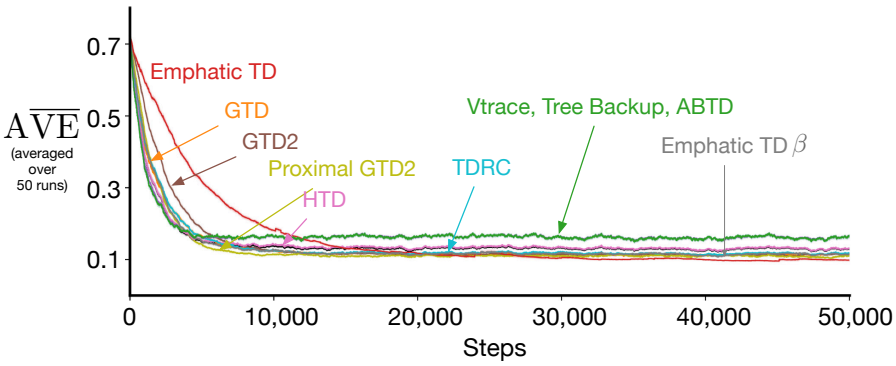


Figure 7.5: Learning curves for the best algorithm instances of each learning algorithm in the Rooms task with general λ. Emphatic TD(λ) learned the slowest, followed by GTD2(λ). V-trace, Tree Backup, and ABTD converged to a statistically significantly worse asymptotic error than other algorithms.

phatic TD(0) had enough time to learn, it would converge to a lower asymptotic solution than other algorithms in the case with $\lambda = 0$, as shown by the straight dashed lines of Figure 7.4

The results from the Collision and Rooms task collectively show that Emphatic TD(λ) tends to have a lower asymptotic error level, but is more prone to the problem variance. On the Collision task, Emphatic TD(λ) had a lower asymptotic error level and learned faster than other algorithms. Moving on to the Rooms task, Emphatic TD learned slower than other algorithms, but still had a lower asymptotic error.

7.5 Gradient-TD Algorithms Applied to the Rooms Task

To study Gradient-TD algorithms in more detail, we set $\lambda = 0$ and analyze the effect of η on performance, where $\alpha = \eta * \alpha_{\mathbf{v}}$, and $\alpha_{\mathbf{v}}$ is the second step size.

Error as a function of α for various values of η is shown in Figure 7.6. Each panel shows the performance of an algorithm as a function of α for four values of η. The errors of algorithm instances with the same η are connected with a line. Each panel shows results of one algorithm in solid lines. Each panel additionally shows the performance of one extra algorithm in dashed lines

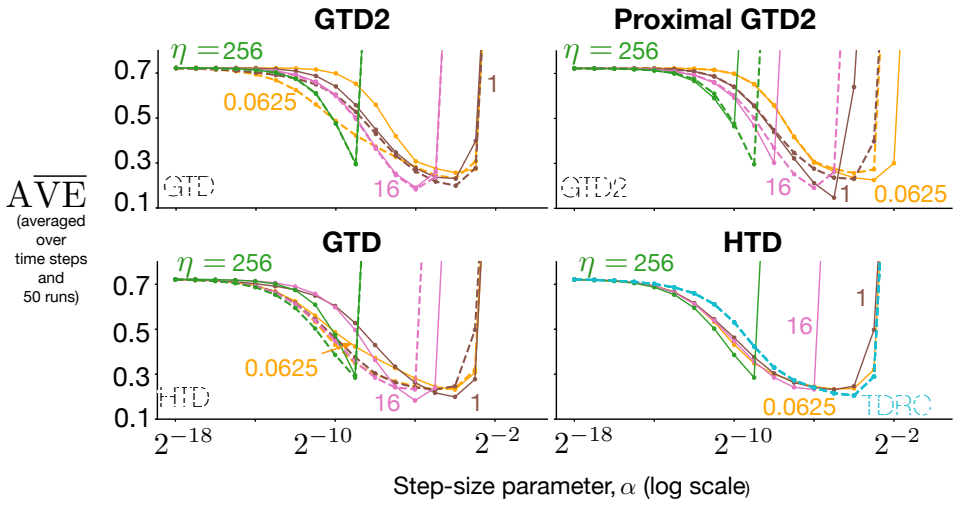


Figure 7.6: Error as a function of α and η at $\lambda = 0$ on the Rooms task. A second algorithm's performance is shown in each panel for comparison. Proximal GTD2 had the lowest error but was more sensitive to α than other algorithms. TDRC and HTD had the lowest sensitivity to α.

for comparison. For example, the upper left panel shows the performance of GTD2(0) in solid and GTD(0) in dashed lines. The dashed lines were always below the solid lines, meaning that GTD(0) had a lower error than GTD2(0) over all parameters. Difference between GTD(0) and GTD2(0) was largest with small η.

With a thorough understanding of one panel, let us now move on to comparing the algorithms in all panels. We first notice that all algorithms solved the problem fairly well. According to the upper right panel, Proximal GTD2 had the lowest error among all algorithms, but only with one of its parameter settings. Proximal GTD2 had a lower error than GTD2 for small η. For larger η the reverse was true. According to the lower left panel, GTD had a slightly lower error than HTD; however, HTD was less sensitive to α, specifically with $\eta = 0.0625$. According to the lower right panel, TDRC's bowl was almost as wide as HTD's widest bowl. TDRC has one tuned parameter and thus has one curve.

Although Proximal GTD2 performed better than others, it did so with only one parameter setting, and thus the improvement it provides is not of much practical importance. HTD, GTD, and TDRC all performed well and

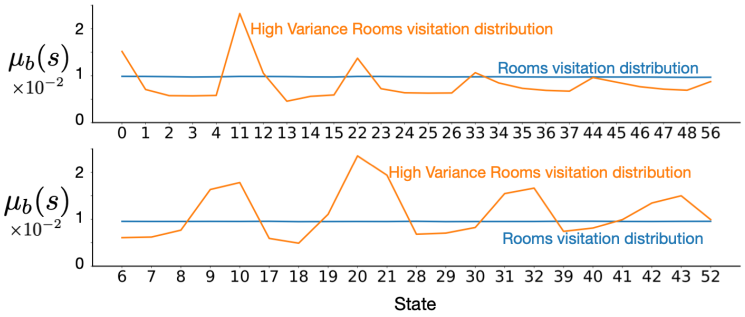


Figure 7.7: State visitation distribution for Rooms (shown in blue) and High Variance Rooms (shown in orange) tasks for two of the sub-tasks that are active in the two lower rooms shown in blue and orange respectively. The numbers on the x-axis are the state numbers with 0 being the bottom left state, and the state immediately to the right of state 0 being state 1. The state immediately above state 0 is state 11. Changing the policy in one of the states resulted in a vastly different state visitation distribution in the two tasks.

were robust to the choice of α. TDRC, specifically, with one tuned parameter, is the easiest to use algorithm for solving the Rooms task. Conclusions made here are similar to the ones made from the Collision experiment.

7.6 High Variance Rooms Task

With a slight modification of the Rooms task, we increased its variance. We changed the behavior policy in four states such that one action is chosen with 0.97 and the three other actions with 0.01 probability. These states are the ones shaded in blue in Figure 7.1. In the two left rooms, the `left` action is chosen with 0.97 probability, and in the two right rooms, the `right` action. This means that, if the `down` action is chosen in the blue state in the upper right room, the importance sampling will be $\frac{1}{1/100}$. The new task is called the High Variance Rooms task. If the agent starts from the upper left state in the upper right room, takes two `right` actions, and then six `down` actions, the product of importance sampling ratios will be $2^{14} \times 25$. In addition to more extreme importance sampling ratios, this small change in the behavior policy largely changes the state visitation distribution compared to the Rooms task. The states to the left of the blue states in the two left rooms, and the states

to the right of the blue states in the two right rooms are visited more often. Visitation distributions are shown in Figure 7.7.

7.7 Experimental Setup

The experimental setup for this task was the same as the Rooms task. Number of time steps, number of runs, and the algorithm instances were all the same.

7.8 Main Results of the High Variance Rooms Experiment

Main results for the High Variance Rooms task are plotted in Figure 7.8. The variance caused by the importance sampling ratio impacted all algorithms except Tree Backup(λ), V-trace(λ), and ABTD(ζ). These three algorithms reached an error of 0.2 (shown as a thin gray line), which was the lowest error achieved on this task. Similar to the Rooms and the Collision tasks, these three algorithms had their best performance with $\lambda = 1$.

Now let us focus on the rest of algorithms in the three first rows of Figure 7.8. All algorithms except Emphatic TD(λ), Proximal GTD2(λ), and GTD2(λ) reached the 0.23 error level (shown as a thin gray line). These three algorithms were sensitive to α and did not perform well. Emphatic TD(λ) reached an error of about 0.45 which was significantly higher than the error achieved by any other algorithm.

On this task, we divide the algorithms into three tiers. The top tier comprises of Tree Backup(λ), V-trace(λ), and ABTD(ζ) whose error was the lowest. The behavior of these was similar across Collision, Rooms, and High Variance Rooms tasks. In the middle tier are Off-policy TD(λ), GTD(λ), HTD(λ), and TDRC(λ). These algorithms achieved a slightly higher error than the top tier algorithms but still reasonably solved the task. The bottom tier comprises of Emphatic TD(λ), GTD2(λ), and Proximal GTD2(λ) whose best error level was even higher than second tier algorithms.

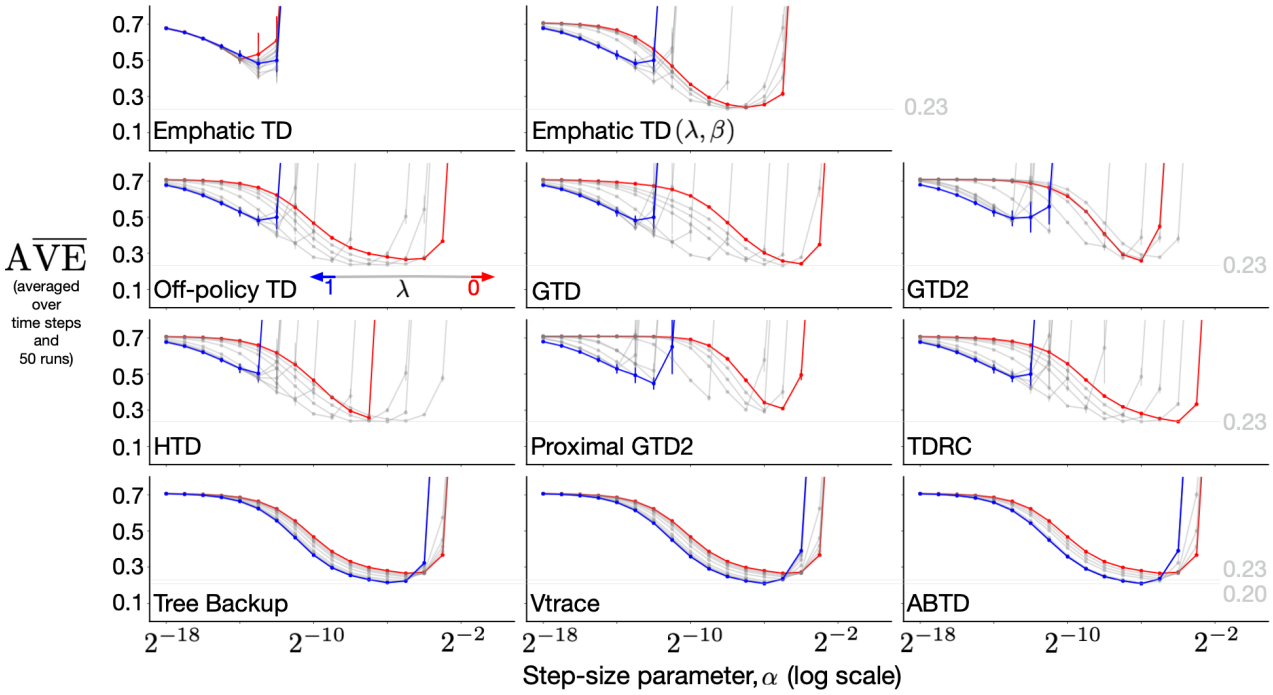


Figure 7.8: Error as a function of α and λ for all algorithms on the High Variance Rooms task. Tree Backup(λ), V-trace(λ), and ABTD(ζ) reached the lowest error level (0.2) and were in the top tier. All other algorithms except for Emphatic TD(λ), Proximal GTD2(λ), and GTD2(λ) reached the 0.23 error-level and were in the middle tier. Emphatic TD(λ), Proximal GTD2(λ), and GTD2(λ) had a higher error than the rest of the algorithms and were more sensitive to α and were grouped into the bottom tier.

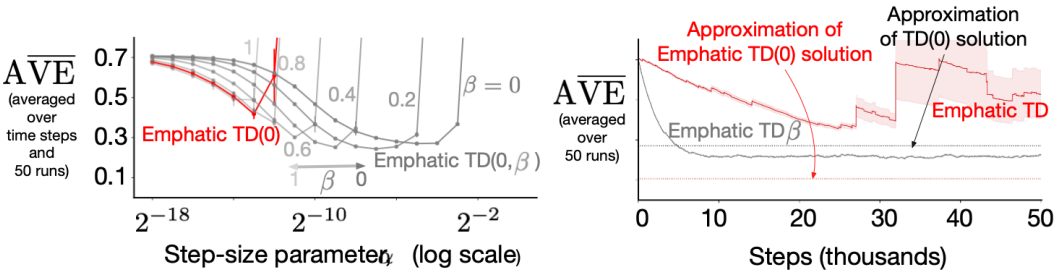


Figure 7.9: Error as a function of α and β at $\lambda = 0$ on the High Variance Rooms task is shown on the left. Two best learning curves for Emphatic TD(λ, β) and Emphatic TD(λ) on the right show that Emphatic TD(λ, β) learned faster.

7.9 Emphatic-TD Algorithms Applied to the High Variance Rooms Task

We now turn to studying the Emphatic-TD algorithms in more detail. We set $\lambda = 0$, and study the behavior of Emphatic TD(λ, β) with varying β. Errors for various values of β are shown in the left panel of Figure 7.9. The best performance was observed with small values of β. The bowl was nice and wide with $\beta = 0$ and $\beta = 0.2$. After that, with increasing β, the error consistently increased.

These results show that varying β significantly improves Emphatic TD(λ, β)'s performance. Without β, Emphatic TD(0) performed quite poorly on the High Variance Rooms task due to large variance. These results and the results from the Rooms task show that β's role becomes more salient as the problem variance increases. On the Collision task, no improvement was observed when varying β. On the Rooms task, intermediate values of β resulted in the best performance, and in the High Variance Rooms task, small values. The trend shows that as the problem variance increases, the magnitude of β that results in the best performance becomes smaller.

Learning curves for best algorithm instances of Emphatic TD(0) and Emphatic TD(0, β) are shown in the right panel of Figure 7.9. These learning curves correspond to algorithm instances that resulted in minimum AUC. Emphatic TD(0, β) learned significantly faster than Emphatic TD. Emphatic

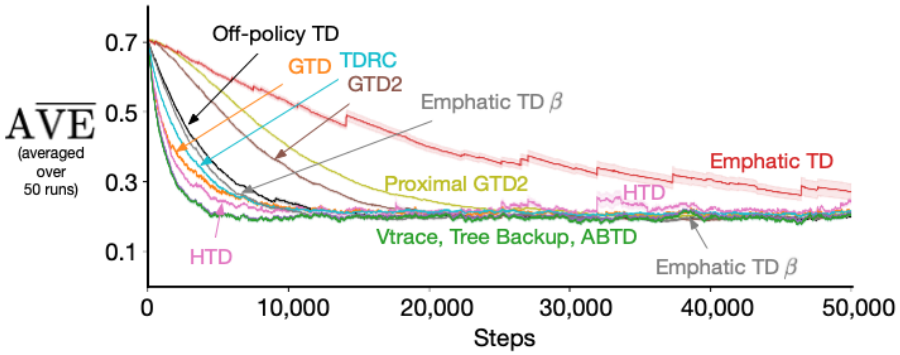


Figure 7.10: Best algorithm instances of each learning algorithm for general λ on the High Variance Rooms task. Emphatic TD learned the slowest, followed by the Proximal GTD2 and GTD2 algorithms. V-trace, Tree Backup, and ABTD learned the fastest.

TD(0) did not learn a reasonable approximation of the value function, probably due to being affected by the problem variance. The two dashed lines in the right panel of the figure show the approximate solutions for Emphatic TD(0) and Off-policy TD(0). Emphatic TD's solution had a significantly smaller $\overline{\text{AVE}}$ than Off-policy TD. This means that if the high variance was not present, Emphatic TD would find a solution with lower error than other algorithms such Off-policy TD(λ).

Learning curves for the best algorithm instances of all learning algorithms for general λ are shown in Figure 7.10. Emphatic TD(λ) learned significantly slower than the rest of the algorithms, followed by Proximal GTD2(λ) and GTD2(λ).

In Rooms, High Variance Rooms, and Collision tasks, Emphatic TD had a lower asymptotic error than other algorithms. This has also been observed in some previous studies (Ghiassian, Rafiee, & Sutton, 2016). However, as the problem variance increases, Emphatic TD(λ) tends to learn slower. On the Collision task, it learned the fastest, on the Rooms task it learned slower than other algorithms, and it failed to learn a good approximation of the value function in the High Variance Rooms task. The trend shows that Emphatic TD has a smaller asymptotic error across tasks but might overall be more prone to the variance issue than other algorithms.

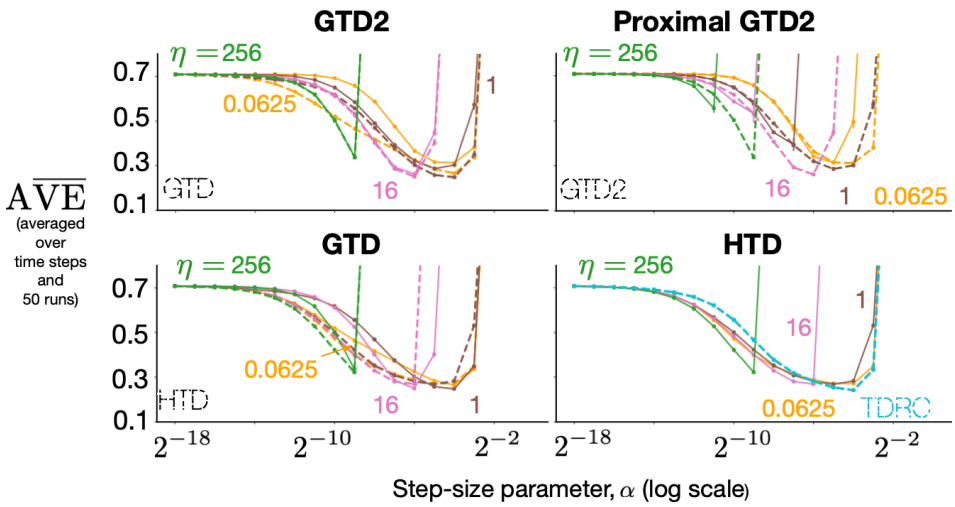


Figure 7.11: Error as a function of α and η at $\lambda = 0$ on the High Variance Rooms task. The error of Proximal GTD2 (solid lines in the upper right panel) was higher than others.

7.10 Gradient-TD Algorithms Applied to the High Variance Rooms Task

We now turn to a more detailed analysis of Gradient-TD algorithms on the High Variance Rooms problem. Errors for $\lambda = 0$ for various η are plotted in Figure 7.11.

Based on the data provided in the upper left panel, GTD2 generally performed worse than GTD. Based on the upper right panel, Proximal GTD2 had a significantly larger error than GTD2. According to the two lower panels, TDRC, HTD, and GTD all performed similarly and were all relatively robust to the choice of α.

Let us now summarize the performance of Gradient-TD algorithms across tasks. On the Collision and Rooms tasks, Proximal GTD2 had the lowest error of all Gradient-TD algorithms. On the High Variance Rooms task, however, it had a higher error than all Gradient-TD algorithms. The trend across problems shows that Proximal GTD2 might be able to reach a lower error level than other Gradient-TD algorithms but is more prone to high variance than other Gradient-TD algorithms. In addition, the lower error level it achieves does not seem to be of much practical utility because it is rare. GTD(λ),

HTD(λ), and TDRC(λ) all seem to work well across tasks. HTD(λ) seems to be easier to tune across problems (see how its various bowls are smoother and wider than GTD in the lower left panel of Figure 7.11). TDRC seems to be the easiest to use Gradient-TD algorithm because it has one tuned parameter and works as well as HTD across tasks.

7.11 Possibility of Larger Empirical Studies

Off-policy learning has been essential to many of the recent successes of deep reinforcement learning. The DQN architecture (Mnih et al., 2015) and its successors such as Double DQN (van Hasselt, Guezm, & Silver, 2016). The core of many of these architectures is Q-learning (Watkins, 1989), the first algorithm developed for off-policy control. Recent research used some modern off-policy algorithms such as V-trace and Emphatic-TD within deep reinforcement learning architectures (Espeholt et al., 2018; Jiang et al., 2021), but it remains unclear which of the many off-policy learning algorithms developed to date empirically outperforms others.

Unfortunately, due to the computational burden, it is not possible to conduct a large comparative study in a complex environment such as the Arcade Learning Environment (ALE). The original DQN agent (Mnih et al., 2015) was trained for one run with a single parameter setting. Of course, the number of runs necessary for an empirical study depends on the distribution of the underlying data and the statistics that one likes to compute, but typically, a detailed comparative study needs at least 30 runs and includes a dozen algorithms, each of which have their own parameters. For example, to compare 10 algorithms on the ALE, each with 100 parameter settings (combinations of step-size parameter, bootstrapping parameter, etc.), for 30 runs, we need 30,000 times more compute than what was used to train the DQN agent on an Atari game. One might think that given the increase in available compute since 2015, such a study might be feasible. Moore's law states that the available compute approximately doubles every two years. That means compared to 2015, eight times more compute is at hand today. Taking this into account,

we still need 30,000/8=3750 times more compute than what was used to train one DQN agent. This is simply not feasible now, or in the foreseeable future.

Let us now examine the possibility of conducting a comparative study in a state-of-the-art domain, similar to Atari, but smaller. MinAtar (Young & Tian, 2019) simplifies the ALE considerably, but presents many of the same challenges. To evaluate the possibility of conducting a comparative study in MinAtar, we compared the training time of two agents. One agent used the original DQN architecture (Mnih et al., 2015), and another used the much smaller neural network architecture of Young and Tian (2019) for training in MinAtar. Both agents were trained for 30,000 frames on an Intel Xeon Gold 6148, 2.4 GHz CPU core. On average, each MinAtar training frame took 0.003 of a second and each ALE training frame took 0.043 of a second. To speed up training, we repeated the same procedure on an NVidia V100SXM2 (16GB memory) GPU. Each MinAtar training frame took 0.0023 of a second and each ALE training frame took 0.0032 of a second. The GPU did a good job speeding up the process that used a large neural network (in the ALE), but did not provide much of a benefit on the smaller neural network used in MinAtar. This means, assuming we have enough GPUs to train on, using MinAtar and ALE will not be that different. Given this data, detailed comparative studies in an environment such as MinAtar are still far out of reach. Note that we do not mean to imply that the experiments conducted on MinAtar in Chapter 5 are not important, but to emphasize that it arguably is not easy (if at all possible) to use the MinAtar environment in a detailed empirical study of a dozen algorithms.

Before we close this chapter with conclusions, we would like to mention that it seems plausible to conduct a careful comparative study on classic reinforcement learning tasks, such as Mountain Car (Moore, 1990). This in fact, seems to be a good future research direction as it will help us understand which of the conclusions made here will scale up to larger, more complex tasks.

7.12 Conclusions

Two of the most central challenges of off-policy learning are stability and slow learning. The stability issue first became evident through Baird's counterexample (Baird, 1995). Since then, Baird's counterexample has been used numerous times to exhibit various algorithms' divergence, and had a part in advances made in algorithms proposed for convergent off-policy learning. In this work, we clearly exhibited the variance challenge of off-policy learning in practice. The tasks introduced in this dissertation can be used to assess the algorithms' capability of learning in a high variance setting.

This study along with the Collision task experiment paints a detailed picture of algorithms' performance as a function of the problem variance. Regarding the interplay of the algorithms' performance and variance of the problem, three main points were shown in this chapter:

1. We showed, for the first time, that Emphatic TD(λ) tends to have a lower asymptotic error level but it is more prone to the high variance issue than other algorithms.
2. We showed, for the first time, that Proximal GTD2(λ) seems to be prone to the variance issue as well, but less so than Emphatic TD(λ).
3. We showed, for the first time, that Tree Backup(λ), V-trace(λ), and ABTD(ζ) are most robust to the problem variance but perform worse than other algorithms on simple problems where high variance is not expected.

Our message for practitioners is to use Tree Backup, V-trace, or ABTD in problems where high variance is expected, and to use Emphatic TD(λ) otherwise.

Chapter 8

Speeding up Emphatic TD(λ)

We set out to propose three new algorithmic ideas in this dissertation. This chapter presents the third algorithmic idea: a step-size adaptation algorithm to increase Emphatic TD(λ)'s learning speed. Remember that Emphatic TD(λ) tends to have lower asymptotic error than other algorithms but it can learn more slowly on problems with high variance. The algorithms presented in this chapter adapt the step-size parameter of Emphatic TD(λ) and significantly increase its learning speed.

Two algorithms are introduced in this chapter: Step-size Ratchet and Soft Step-size Ratchet. The main idea behind both algorithms is the same: keep the step-size parameter as large as possible, and ratchet it down when there is a possibility of overshoot. To show that the new algorithms are effective, we combine them with Emphatic TD(λ) and apply them to the Collision, Rooms, and High Variance Rooms tasks. Remember from the previous chapter that Emphatic TD(λ) with constant step-sizes could not learn a good approximation of the value function in the High Variance Rooms task and learned slowly on the Rooms task. In this chapter, we show that, not only the combination of Emphatic TD(λ) and Ratchet algorithms learns a good approximation of the value function on all three problems, but also we show that Ratchet algorithms learn faster than other step-size adaptation algorithms such as Adam, when combined with Emphatic TD(λ).

8.1 Emphatic TD(λ) + Step-size Ratchet

In this section we introduce the first step-size adaptation algorithm and combine it with Emphatic TD(λ). The main idea is to choose at each time step a step-size parameter that is as large as possible (or a fraction of it) without overshooting the target. Depending on how close the target of the update is to the current estimation of the value function, the step-size parameter is reduced only when necessary. The step-size parameter is never increased during learning.

At each time step, we compute the magnitude of the step-size parameter that if used, will result in the estimate of the value function being equal to the target of the update. This means we want to find the step-size that if used, the following equation would hold:

$$\mathbf{w}_{t+1}{}^{\top}\mathbf{x}_t = R_{t+1} + \gamma_{t+1}\mathbf{w}_t^{\top}\mathbf{x}_{t+1}. \tag{8.1}$$

In TD style algorithms, the target of the update is the reward plus the discounted value of the next state. The update rules for Emphatic TD(λ) are:

$$\delta_t \stackrel{\text{def}}{=} \rho_t(R_{t+1} + \gamma_{t+1}\mathbf{w}_t^{\top}\mathbf{x}_{t+1} - \mathbf{w}_t^{\top}\mathbf{x}_t) \tag{8.2}$$

$$F_t \leftarrow \rho_{t-1}\gamma_t F_{t-1} + I_t \quad \text{with } F_0 = I_0$$

$$M_t \stackrel{\text{def}}{=} \lambda_t I_t + (1-\lambda_t)F_t$$

$$\mathbf{z}_t \leftarrow \rho_{t-1}\gamma_t\lambda\mathbf{z}_{t-1} + M_t\mathbf{x}_t \quad \text{with } \mathbf{z}_{-1} = \mathbf{0} \tag{8.3}$$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha_t\delta_t\mathbf{z}_t. \tag{8.4}$$

We now replace $\mathbf{w}_{t+1}$ on the left hand side of (8.1) with the update rule provided for $\mathbf{w}_{t+1}$ in (8.4) and solve for α_t:

$$\begin{aligned}(\mathbf{w}_t + \alpha_t\delta_t\mathbf{z}_t)^{\top}\mathbf{x}_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^{\top}\mathbf{x}_{t+1}\\ \alpha_t\delta_t\mathbf{z}_t^{\top}\mathbf{x}_t &= \underbrace{R_{t+1} + \gamma_{t+1}\mathbf{w}_t^{\top}\mathbf{x}_{t+1} - \mathbf{w}_t^{\top}\mathbf{x}_t}_{\delta_t}\\ &\triangleright \alpha_t = \frac{1}{\mathbf{z}_t^{\top}\mathbf{x}_t},\end{aligned} \tag{8.5}$$

which in the case of full bootstrapping reduces to:

$$\alpha_t = \frac{1}{M_t\mathbf{x}_t^{\top}\mathbf{x}_t}, \tag{8.6}$$

because if $\lambda = 0$, we can see from (8.3) that $\mathbf{z}_t \leftarrow M_t \mathbf{x}_t$.

We refer to the α_t that has the magnitude computed above in (8.5), as the *normalized step-size* parameter. We use a parameter κ, to take a fraction of a normalized step towards the target at each time step. For $\lambda > 0$:

$$\alpha_t = \frac{\kappa}{\mathbf{z}_t^\top \mathbf{x}_t},$$

and for $\lambda = 0$:

$$\alpha_t = \frac{\kappa}{M_t \mathbf{x}^\top \mathbf{x}_t}.$$

It makes intuitive sense for the step-size parameter to be large at the beginning of learning and shrink down as learning goes on. The following minimization at each time step assures that the step-size parameter shrinks or remains the same:

$$\alpha_t \leftarrow \min\left(\alpha_{t-1}, \frac{\kappa}{\mathbf{z}_t^\top \mathbf{x}_t}\right). \tag{8.7}$$

At the first time step, α_{t-1} is set to a large number, maybe ∞. This completes the specification of the Step-size Ratchet algorithm. The following update rules fully specify Emphatic TD(λ) augmented with Step-size Ratchet:

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} \rho_t(R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t) \\
F_t &\leftarrow \rho_{t-1}\gamma_t F_{t-1} + I_t \quad \text{with } F_{-1} = 0 \\
M_t &\stackrel{\text{def}}{=} \lambda_t I_t + (1 - \lambda_t) F_t \\
\mathbf{z}_t &\leftarrow \rho_{t-1}\gamma_t \lambda \mathbf{z}_{t-1} + M_t \mathbf{x}_t \quad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\alpha_t &\leftarrow \min\left(\alpha_{t-1}, \frac{\kappa}{\mathbf{z}_t^\top \mathbf{x}_t}\right) \qquad \text{with } \alpha_{-1} = \infty \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha_t \delta_t \mathbf{z}_t \qquad \text{with } \mathbf{w}_0 = \mathbf{0}, \text{or arbitrary.}
\end{aligned}$$

8.2 Emphatic TD(λ) + Soft Step-size Ratchet

In this section, we propose a new algorithm called Soft Step-size Ratchet, and combine it with Emphatic TD(λ). The goal of Soft Step-size Ratchet is to relax the strict requirement of the Step-size Ratchet algorithm that the step-size parameter can only become smaller or remain the same size at each time step. We will show that we are able to increase the magnitude of the step-size

parameter at times, while maintaining the main idea of the Step-size Ratchet algorithm. We show that this change will result in increasing the learning speed.

The Soft Step-size Ratchet algorithm works as follows. At each time step, first the step-size parameter is computed using (8.7); the same update rule used for Step-size Ratchet. Second, the step-size parameter, α_t, is used to update the weight vector $\mathbf{w}_t$; again the same as what the Step-size Ratchet algorithm does. The Step-size Ratchet algorithm, at this point will go back to the first step and continue from there. The Soft Step-size Ratchet algorithm, instead, uses α_t and α_{t-1} to compute a new α_{t-1} that will be used in the next round of the updates once the execution of the algorithm goes back to the first step. This means that the only difference between the Step-size Ratchet and Soft Step-size Ratchet algorithms is that the α_{t-1} used in (8.7) will be updated before moving on to the next time step, pretending that the previous step-size parameter was in fact larger than what was used to update $\mathbf{w}$. The update rule we use for computing the new α_{t-1} is:

$$\alpha_{t-1} \leftarrow \alpha_t + (\alpha_{t-1} - \alpha_t) \times \tau, \tag{8.8}$$

where τ is a tunable parameter in (0, 1), that we set to 0.5 in all our experiments. After executing (8.8), the agent moves on to the next time step, and goes back to the first step and continues from there.

Let us now examine (8.8) more closely. Remember that before executing (8.8), the step-size parameter, α_t, is calculated using (8.7) at each time step, from which it immediately follows that $\alpha_{t-1} \geq \alpha_t$ for $\forall t$. It then follows that the term $(\alpha_{t-1} - \alpha_t)$ in (8.8) is always greater than or equal to zero. Let us first consider the case when it is zero. The difference being zero means $\alpha_t = \alpha_{t-1}$, meaning that α_{t-1} will not change after executing (8.8) because the term $(\alpha_{t-1} - \alpha_t)$ is equal to zero. If the difference is not zero, it means $\alpha_{t-1} > \alpha_t$, in which case the value of α_{t-1} after performing update (8.8) will become larger proportional to τ. Let us for example assume $\alpha_{t-1} = 1$, and the new step-size parameter computed by (8.7) is $\alpha_t = 0.5$, and $\tau = 0.5$. In this

case, the α_{t-1} for the next round of updates will be:

$$\alpha_{t-1} = 0.5 + (1 - 0.5) \times 0.5 = 0.75$$

The magnitude of increase in α_{t-1} is proportional to the difference between α_{t-1} and α_t and is also proportional to the magnitude of τ. At the first time step, we set $\alpha_{-1} = \frac{\kappa}{\mathbf{z}_t^\top \mathbf{x}_t}$. We call this new algorithm *Soft Step-size Ratchet* because ratcheting down the step-size parameter is soft, in that the step-size parameter can sometimes become a little larger. The following update rules fully specify Emphatic TD(λ) augmented with Soft Step-size Ratchet:

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} \rho_t(R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t) \\
F_t &\leftarrow \rho_{t-1}\gamma_t F_{t-1} + I_t \quad \text{with } F_{-1} = 0 \\
M_t &\stackrel{\text{def}}{=} \lambda_t I_t + (1 - \lambda_t) F_t \\
\mathbf{z}_t &\leftarrow \rho_{t-1}\gamma_t \lambda \mathbf{z}_{t-1} + M_t \mathbf{x}_t \quad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\alpha_t &\leftarrow \min\left(\alpha_{t-1}, \frac{\kappa}{\mathbf{z}_t^\top \mathbf{x}_t}\right) \qquad \text{with } \alpha_{-1} = \frac{\kappa}{\mathbf{z}_t^\top \mathbf{x}_t} \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha_t \delta_t \mathbf{z}_t \qquad \text{with } \mathbf{w}_0 = \mathbf{0} \\
\alpha_{t-1} &\leftarrow \alpha_t + (\alpha_{t-1} - \alpha_t) \times \tau.
\end{aligned}$$

8.3 Emphatic TD(λ) + Adam

The adaptive moment estimation algorithm (Adam), also known as the Adam optimizer, is one of the most commonly used step-size adaptation algorithms used in deep reinforcement learning (Kingma & Ba, 2017). Adam is often used with neural network function approximation to increase learning speed. Adam uses statistics from the gradient vector to compute the direction and size of the update, in each dimension of the space, at each time step. The most significant difference between Adam and the Ratchet algorithms is that Adam computes a vector of step-sizes at each time step, one step-size scalar for each element of the weight vector, whereas the Ratchet algorithms compute a universal step-size parameter at each time step, one scalar step-size for all elements of $\mathbf{w}$. As a baseline in our experiments, we used Emphatic TD(λ) combined with Adam. Of course other step-size adaptation algorithms, such

as AdaGrad (Duchi, Hazan, & Singer, 2011) or AdaGain (Jacobsen et al., 2019) can be included as baseline algorithms. We chose to include Adam in our experiments rather than other algorithms simply because it is shown to be effective on a variety of problems.

Combining Emphatic TD(λ) and Adam is simple. Adam typically operates on the gradient vector. Here, instead of the gradient vector, we apply Adam to the directions suggested by Emphatic TD(λ). The update rules for the combination of Emphatic TD(λ) and Adam are as follows:

$$
\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} \rho_t(R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t) \\
F_t &\leftarrow \rho_{t-1}\gamma_t F_{t-1} + I_t \quad \text{with } F_{-1} = 0 \\
M_t &\stackrel{\text{def}}{=} \lambda_t I_t + (1-\lambda_t) F_t \\
\mathbf{z}_t &\leftarrow \rho_{t-1}\gamma_t \lambda \mathbf{z}_{t-1} + M_t \mathbf{x}_t \quad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{g_t} &\stackrel{\text{def}}{=} \delta_t \mathbf{z}_t \qquad (\text{set } \mathbf{g_t} \text{ to the direction computed by Emphatic TD}(\lambda)) \\
\mathbf{m_{t+1}} &\leftarrow \beta_1 \mathbf{m_t} + (\mathbf{1} - \beta_1)\mathbf{g_t} \quad \text{with } \mathbf{m_0} = \mathbf{0} \\
\mathbf{v_{t+1}} &\leftarrow \beta_2 \mathbf{v_t} + (\mathbf{1} - \beta_2)\mathbf{g_t^2} \quad \text{with } \mathbf{v_0} = \mathbf{0} \\
\widehat{\mathbf{m}}_{\mathbf{t+1}} &\stackrel{\text{def}}{=} \mathbf{m_{t+1}}/(\mathbf{1} - \beta_1^{\mathbf{t+1}}) \\
\widehat{\mathbf{v}}_{\mathbf{t+1}} &\stackrel{\text{def}}{=} \mathbf{v_{t+1}}/(\mathbf{1} - \beta_2^{\mathbf{t+1}}) \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \widehat{\mathbf{m}}_{\mathbf{t+1}}/(\sqrt{\widehat{\mathbf{v}}_{\mathbf{t+1}}} + \epsilon) \quad \text{with } \mathbf{w_0} = \mathbf{0}.
\end{aligned}
$$

These update rules specify the direction of the update in each dimension. The rest of the update rules, specify the magnitude of the update in each direction in the space. The variable $\mathbf{m_t}$ is the biased first moment estimate, $\mathbf{v_t}$ is the biased second raw moment estimate, $\widehat{\mathbf{m}}_{\mathbf{t+1}}$ is the bias-corrected first moment estimate, $\widehat{\mathbf{v}}_{\mathbf{t+1}}$ is the bias-corrected second raw moment estimate, and ϵ is a small constant. Adam has four free parameters: β_1 and β_2 that determine the weighting of the average of the first and second moments, α, which is the step-size parameter, and finally ϵ that prevents division by zero.

8.4 Emphatic TD(λ) + AlphaBound

The AlphaBound Algorithm (Dabney & Barto, 2012) is similar to the Step-size Ratchet algorithm in that it has a single step-size parameter (rather than a vector of step-size parameters one for each direction) that only decreases in value over the course of learning. The schedule by which the step-size parameter shrinks is different from the schedule used by the Step-size Ratchet algorithm. The intuition behind the AlphaBound algorithm is to make sure that the TD-error at time step t gets closer to 0 after each update. This means that AlphaBound assures $|\delta'_t| < |\delta_t|$ at each time step, where δ'_t is the TD-error after a single update to the weight vector. In fact, it is shown by Dabney and Barto (2012) that AlphaBound guarantees the condition $|\delta'_t| < |\delta_t|$ is satisfied at each time step by using the following update rule:

$$\alpha_t \leftarrow \min\left(\alpha_{t-1}, \frac{1}{|\ \mathbf{z}_t^\top(\gamma\mathbf{x}_{t+1} - \mathbf{x}_t)\ |}\right), \qquad \text{(whenever } \mathbf{z}_t^\top(\gamma\mathbf{x}_{t+1} - \mathbf{x}_t) < 0\text{)}$$

with α_0 being the free parameter set by the user. It is shown by Dabney and Barto (2012) that the $|\delta'_t| < |\delta_t|$ inequality and the requirement that $\alpha \in [0, 1]$ forces $\alpha = 0$ whenever $\mathbf{z}_t^\top(\gamma\mathbf{x}_{t+1} - \mathbf{x}_t) > 0$, which is equivalent to ignoring the update completely. Dabney and Barto (2012) suggests to set the step-size parameter equal to 1 at the first time step, meaning $\alpha_0 = 1$. The AlphaBound algorithm was originally applied to the TD(λ) algorithm. Here, we apply it to the Emphatic TD(λ) algorithm. When applied to Emphatic TD(λ), the AlphaBound algorithm remains the same as when it is applied to TD(λ). The only difference between the two cases is the eligibility trace vector, $\mathbf{z}_t$ which includes the emphasis when Emphatic TD(λ) is used. For completeness, we provide the update rules that fully specify the Emphatic TD(λ) algorithm

augmented with AlphaBound:

$$
\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} \rho_t(R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t) \\
F_t &\leftarrow \rho_{t-1}\gamma_t F_{t-1} + I_t \quad \text{with } F_{-1} = 0 \\
M_t &\stackrel{\text{def}}{=} \lambda_t I_t + (1-\lambda_t) F_t \\
\mathbf{z}_t &\leftarrow \rho_{t-1}\gamma_t \lambda \mathbf{z}_{t-1} + M_t \mathbf{x}_t \quad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\alpha_t &\leftarrow \min\left(\alpha_{t-1}, \frac{1}{|\ \mathbf{z}_t^\top(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t)\ |}\right) \qquad (\text{if } \mathbf{z}_t^\top(\gamma \mathbf{x}_{t+1} - \mathbf{x}_t) < 0) \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha_t \delta_t \mathbf{z}_t \qquad \text{with } \mathbf{w}_0 = \mathbf{0}.
\end{aligned}
$$

8.5 The Collision Task Experiment

We applied the following combinations of algorithms to the Collision task: Emphatic TD(λ) and Step-size Ratchet, Emphatic TD(λ) and Soft Step-size Ratchet, Emphatic TD(λ) and AlphaBound, Emphatic TD(λ) and Adam, and finally Emphatic TD(λ) with a constant step-size parameter. Often, we refer to these combinations by the step-size adaptation algorithm name, leaving out Emphatic TD(λ) from the name because it is common across all combinations.

All of the experimental setup remains the same as what was discussed in Chapter 6 other than the parameters of the step-size adaptation algorithms, which we will discuss shortly. The Collision task in conjunction with its behavior policy, was used to generate 20,000 time steps, and 50 independent runs. Each run, used a different random binary feature representation. The feature representations were the same as the ones used previously in Chapter 6.

Similar to the previous chapter, we use the term algorithm instance to refer to an algorithm with a specific set of parameters. We applied many algorithm instances to the Collision task. The parameters of algorithms included all combinations of: two values of λ (0, 0.9), 19 values of α for constant step-size parameters ($\alpha = 2^{-x}$ where $x \in \{0, 1, 2, \ldots, 17, 18\}$), 19 values of α_0 for AlphaBound and Adam algorithms ($\alpha_0 = 2^{-x}$ where $x \in \{0, 1, 2, \ldots, 17, 18\}$), 19 values of κ for Step-size Ratchet and Soft Step-size Ratchet ($\kappa = 2^{-x}$ where $x \in \{0, 1, 2, \ldots, 17, 18\}$), one value of τ for Soft Step-size Ratchet ($\tau = 0.5$), three values of β_1 for Adam (0.9, 0.99, 0.999), and three values of β_2 for Adam

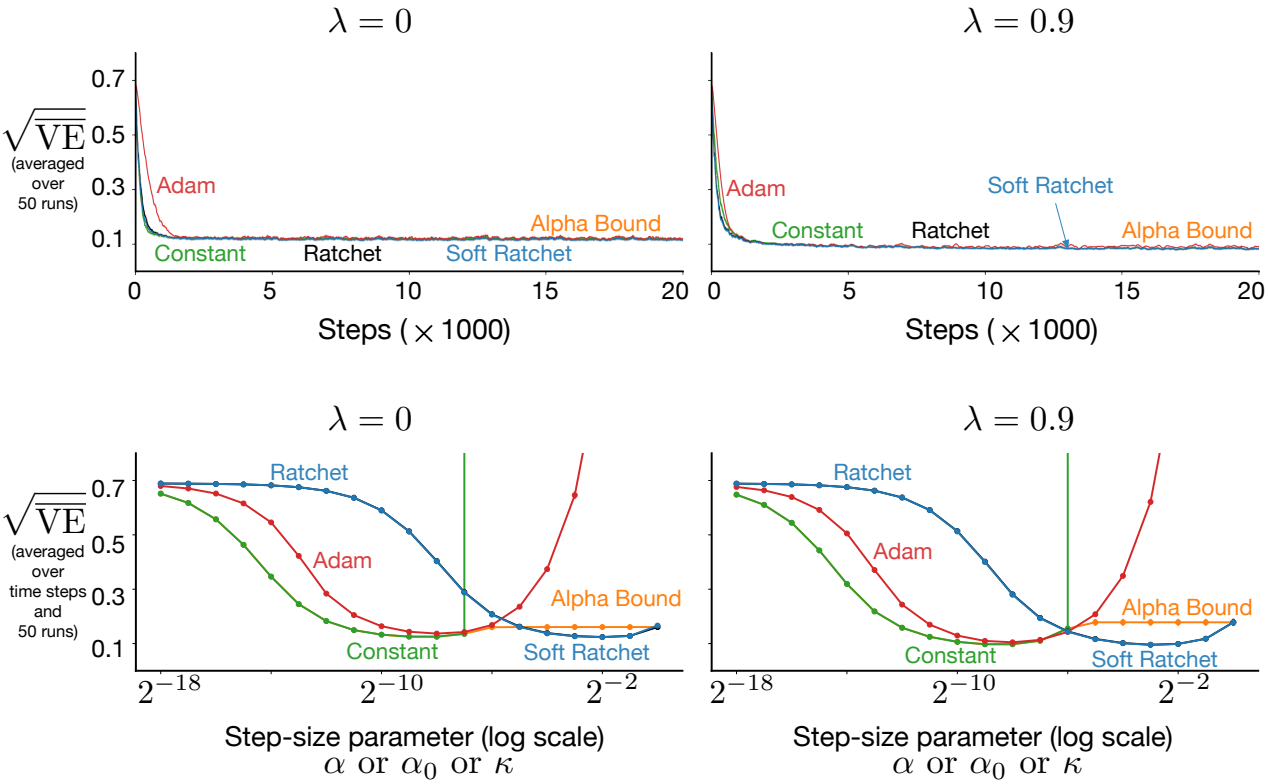


Figure 8.1: Results of applying the combination of Emphatic TD(λ) and one of the Step-size Ratchet, Soft Step-size Ratchet, Adam, AlphaBound, or constant step sizes to the Collision task. The first row shows the learning curves and the second row shows the parameter sensitivity curves. All algorithms learned with a similar speed, with the exception of Adam in the full bootstrapping case that learned more slowly than other algorithms. Step-size Ratchet and Soft Step-size Ratchet algorithms had their U-shaped bowl shifted to the right and had their minimum at around 2^{-2}.

(0.9, 0.99, 0.999). Finally, for Adam we set $\epsilon = 10^{-8}$, as suggested in the original paper.

Similar to the original Collision experiment, at the beginning of each run, the weight vector was initialized to $\mathbf{w}_0 = \mathbf{0}$, and then was updated at each time step by the algorithm instance to produce a sequence of $\mathbf{w}_t$. At each time step, we recorded the $\overline{\text{VE}}(\mathbf{w}_t)$ and used it to plot the results we discuss in the next section.

8.6 Collision Task Results

The results are shown in Figure 8.1. The top row shows learning curves over 20,000 time steps and the bottom row shows parameter sensitivity curves. Results for two values of the bootstrapping parameter are shown: $\lambda = 0$ (full bootstrapping) in the left column, and $\lambda = 0.9$ (minimal bootstrapping) in the right column. For Adam that had more than one parameter, to plot one sensitivity curve, we first found the combination of all parameters that resulted in the best overall performance (minimum area under the learning curve), then fixed all parameters except the step-size parameter, and plotted the performance over the step-size parameter with the rest of the parameters fixed.

The learning curves show that compared to when constant step sizes were used, no improvement was observed when step-size adaptation algorithms were used. In fact, in one case, with $\lambda = 0$, Adam resulted in a worse performance than when constant step sizes were used. Most learning curves in the figures are on top of each other and thus are not visible.

The parameter sensitivity plots show that all algorithms had a U-shaped curve of a similar width, meaning that, at logarithmic scale, all algorithms were similarly sensitive to the choice of their parameter. The U-shaped curve for Step-size Ratchet and Soft Step-size Ratchet algorithm were shifted to the right meaning that at linear scale, Step-size Ratchet and Soft Step-size Ratchet were less sensitive to the choice of the step-size parameter than other algorithms.

The standard error for each algorithm is shown as a shaded area around the learning curves and as error bars on the sensitivity curves. In almost all cases, the error bars are not visible due to being small. In the case with $\lambda = 0$, the combination of Adam and Emphatic TD learned statistically significantly slower than other algorithms. Other than that, no statistically significant difference was observed between the algorithms on the Collision task. We conclude that none of the algorithms we considered here improves the performance of Emphatic TD(λ) when applied to the Collision task.

8.7 Rooms Task Experiment

We applied the same set of algorithms to the Rooms task. The experimental setup remained the same as the original Rooms experiment discussed in Chapter 7. In short, we applied the algorithms to the task for 50,000 time steps and 50 independent runs. We used tile coding to create a feature representation for each state. The range of parameters applied to the Rooms task were the same as the ones applied to the Collision task, discussed in Section 8.5 of this chapter.

8.8 Rooms Task Results

The results are shown in Figure 8.2. The top row shows the learning curves, and the bottom row shows the parameter sensitivity curves.

Let us first focus on the learning curve for the full bootstrapping case shown on the upper left panel of Figure 8.2. Unlike the Collision task where the step-size adaptation algorithms did not have any positive effect, on the Rooms task, we see the positive effect of applying the step-size adaptation algorithms to Emphatic TD(λ) clearly. Emphatic TD(λ) with constant step sizes learned the slowest, followed by the AlphaBound algorithm and then Adam. Emphatic TD(0) combined with the Step-size Ratchet algorithm learned faster than constant, AlphaBound, and Adam, as shown in the upper left panel of Figure 8.2. The Soft Step-size Ratchet algorithm learned the fastest. In the case of full bootstrapping, the Soft Step-size Ratchet algorithm converged to the lowest asymptotic error level, followed by the Step-size Ratchet algorithm. The third lowest asymptotic error level was achieved with constant step sizes, followed by AlphaBound. Interestingly, the worst asymptotic error level was observed when Adam was used.

Let us now move on to consider the learning curves for the minimal bootstrapping case shown in the upper right panel of Figure 8.2. Similar to the full bootstrapping case, the Soft Step-size Ratchet algorithm was the fastest, followed by the Step-size Ratchet algorithm. These were followed by Adam,

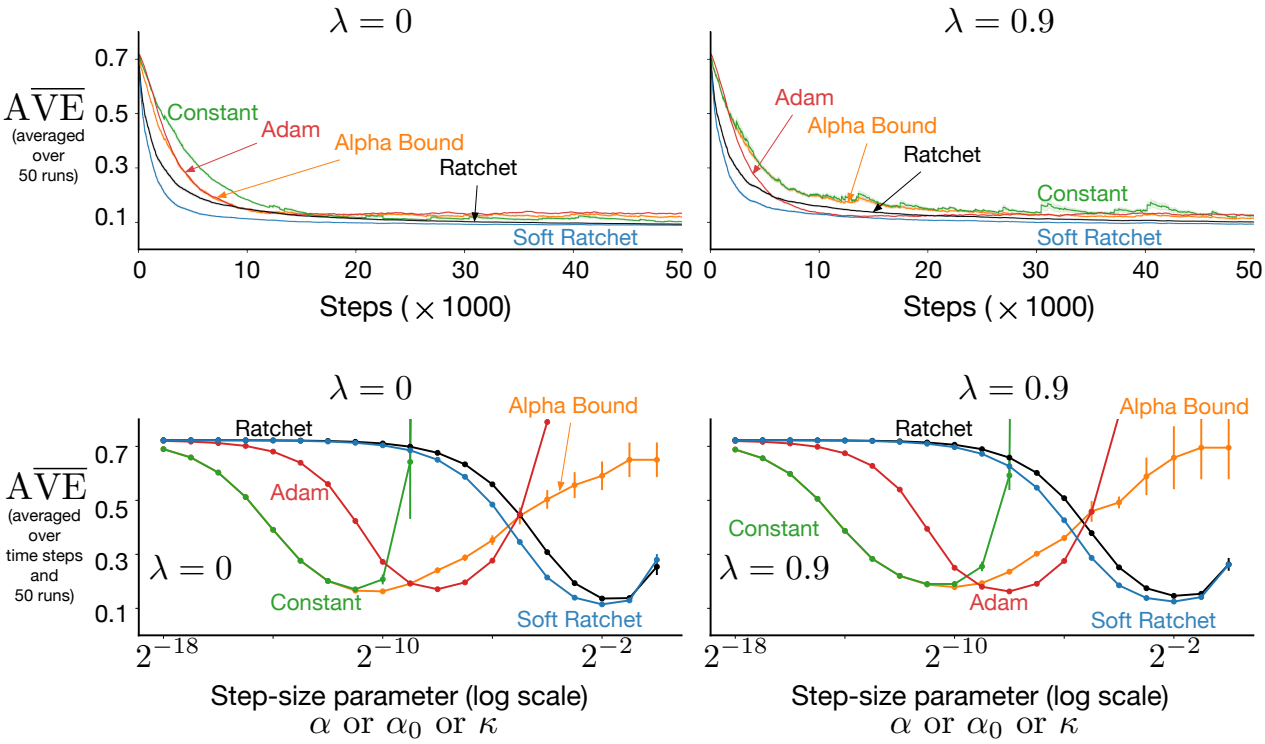


Figure 8.2: Results of applying the combination of Emphatic TD(λ) and one of the Step-size Ratchet, Soft Step-size Ratchet, Adam, AlphaBound, or constant step sizes to the Rooms task. The first row shows the learning curves and the second row shows the parameter sensitivity curves. The Soft Step-size Ratchet algorithm learned the fastest and converged to the lowest error level followed by the Step-size Ratchet algorithm. Step-size Ratchet and Soft Step-size Ratchet algorithms had their U-shaped bowl shifted to the right compared to other algorithms. They had their minimum at around 2^{-2}.

and then AlphaBound and finally constant step size parameters. Regarding asymptotic error level, the results were similar to the full bootstrapping case. The Soft Step-size Ratchet converged to the lowest error level followed by the Step-size Ratchet algorithm. The asymptotic error level was similar when the step-size parameter was constant and when Adam was used. With the AlphaBound algorithm, the asymptotic error level was a little lower than constant, but statistically significantly higher than Soft Step-size Ratchet and Step-size Ratchet algorithms.

Two parameter sensitivity plots are shown at the lower row of Figure 8.2. Similar to what was observed on the Collision task, the sweet spot for the Soft

Step-size Ratchet and Step-size Ratchet algorithms were shifted to the right compared to other algorithms. These two algorithms had the largest sweet spot at linear scale.

Overall, Step-size Ratchet and Soft Step-size Ratchet seem to be effective in increasing the learning speed of Emphatic TD(λ) when applied to the Rooms task. Moreover, at linear scale, they seem to have a larger sweet spot than other algorithms. Another interesting fact is that the range of the best parameter for Soft Step-size Ratchet and Step-size Ratchet algorithms is around 2^{-2} for both the Collision and Rooms tasks. To see if this remains true on other tasks, more experimental results are required.

8.9 High Variance Rooms Task Experiment

We applied the same set of algorithms to the High Variance Rooms task. The experimental setup remained the same as what we discussed in Chapter 7. The algorithms were applied to the task for 50,000 time steps and 50 independent runs. The algorithm instances and range of parameters applied to the task remained the same as the ones used in the Collision and Rooms tasks in this chapter.

8.10 High Variance Rooms Task Results

The results are plotted in Figure 8.3. Similar to the previous experiments, the top row shows the best learning curves, and the bottom row shows the parameter sensitivity curves. Let us first focus on the two learning curves for full and minimal bootstrapping. As we see, the difference between the algorithms is more nuanced in the High Variance Rooms task than the Rooms task. Emphatic TD(λ) with a constant step-size parameter had difficulty learning the value function due to the high variance of the updates. Similar to the Rooms task, the Soft Step-size Ratchet algorithm converged the fastest and to the lowest asymptotic error than all other algorithms. The Step-size Ratchet algorithm was the second fastest algorithm during the first 8000 time steps or so. The Adam algorithm started learning slower than the Step-size Ratchet

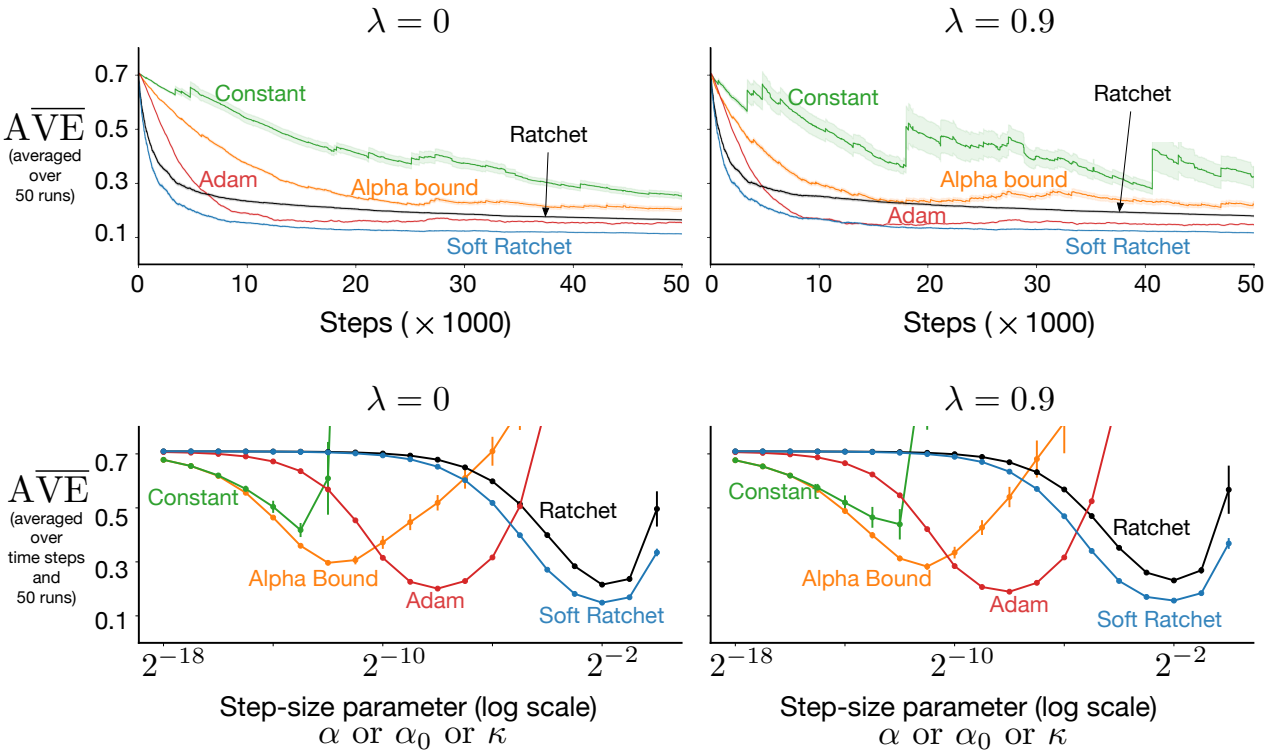


Figure 8.3: Results of applying the combination of Emphatic TD(λ) and one of Step-size Ratchet, Soft Step-size Ratchet, Adam, AlphaBound, or constant step sizes to the High Variance Rooms task. The first row shows the learning curves and the second row shows the parameter sensitivity curves. The Soft Step-size Ratchet algorithm learned the fastest followed by the Step-size Ratchet algorithm. Adam, on the other hand converged faster than the Step-size Ratchet algorithm. Step-size Ratchet and Soft Step-size Ratchet algorithms had their U-shaped bowl shifted to the right compared to other algorithms. They had their minimum at around 2^{-2}.

algorithm, but showed an advantage over the Step-size Ratchet algorithm after that. Towards the end of the learning, Adam converged to a lower error level than the Step-size Ratchet algorithm. The AlphaBound algorithm improved upon using constant step sizes, but its learning speed was slower and its asymptotic error level was higher than Adam, Step-size Ratchet, and Soft Step-size Ratchet algorithms.

The parameter sensitivity curves show that all algorithms (except when constant step sizes were used) were equally sensitive to the value of their parameter at logarithmic scale. The Soft Step-size Ratchet algorithm had a

lower error over a range of parameters when compared to other algorithms, followed by Adam, and the Step-size Ratchet algorithm.

Overall, the Step-size Ratchet and Soft Step-size Ratchet algorithms were quite effective on the Rooms and High Variance Rooms tasks, while maintaining the good performance of Emphatic TD(λ) on the Collision task. The Soft Step-size Ratchet algorithm seem to be the best algorithm across tasks consistently. On all tasks, the κ parameter that resulted in the lowest error for Step-size Ratchet and Soft Step-size Ratchet algorithms was around 2^{-2}. It is natural to ask if it is easier to tune the parameter for Step-size Ratchet and Soft Step-size Ratchet than the parameters of other algorithms across tasks? Given the data that we have from the three tasks, it seems like the answer to this question might be yes. To answer this question definitively, more experimental results are of course necessary.

8.11 Conclusions

In this chapter, we proposed two new approaches to adapt the step-size parameter during learning. We found that both approaches are quite effective in speeding up Emphatic TD(λ)'s learning on the Rooms and High Variance Rooms tasks. Although only applied to the Emphatic TD, it will probably be straightforward to apply these two step-size adaptation algorithms to other reinforcement learning algorithms such as the Gradient-TD methods.

In previous chapters, we showed that Emphatic TD tends to have a lower asymptotic error than other algorithms. This was shown through using Least-squares Emphatic TD(λ). We observed that the online incremental version of the Emphatic TD(λ) with constant step size parameters fails to converge to the solution found by Least-squares Emphatic TD(λ) within 50,000 time steps due to high variance. In this chapter, we showed that when online incremental Emphatic TD(λ) is combined with Ratchet algorithms, the combination converges to a low asymptotic error level within 50,000 time steps. Additionally, we showed that, with the two Ratchet algorithms the range of appropriate step-size parameters is larger than other algorithms at linear scale and finding

the sweet spot of the step-size parameters is easy across problems.

8.12 Limitations

The primary goal of this chapter was to introduce the Step-size Ratchet and Soft Step-size Ratchet algorithms and show that they can be useful in speeding up Emphatic TD(λ). The study of the two algorithms proposed in this chapter is in various ways limited.

One limitation of the current study is that it limits the application of Step-size Ratchet and Soft Step-size Ratchet algorithms to increasing the learning speed of Emphatic TD(λ). Not only do we expect the Step-size Ratchet and Soft Step-size Ratchet algorithms to be applicable to other reinforcement learning algorithms, but also we consider this a fruitful future research direction.

Another limitation of this work is the small number of comparisons made with other step-size adaptation algorithms. Comparisons with AdaGain (Jacobsen et al, 2019), AdaGrad (Duchi, Hazan, & Singer, 2011), and newer versions of the Adam including AMSGrad (Reddi, Kale, & Kumar, 2018) would make the results of this chapter significantly stronger. Of course, experiments on new environments are necessary before a trend of results can be established and a final judgement about the merits of the Step-size Ratchet and Soft Step-size Ratchet algorithms can be made.

One other limitation of the Soft Step-size Ratchet algorithm is that is has more than one tuned parameter. However, the experiments show that it might not be that hard for the user to select the τ parameter, because we only used one value of τ in our experiments, and with that one value, the Soft Step-size Ratchet algorithm performed well across tasks. It remains unclear how sensitive the Soft Step-size Ratchet algorithm is to the choice of the τ parameter and if tuning τ can provide a significant performance gain.

Another limitation of both the Step-size Ratchet and Soft Step-size Ratchet algorithms is that they are not readily applicable to non-stationary problems. These algorithms ratchet down the step-size parameter over time, which means if the environment changes during learning, it might be hard for the agent to

adapt to the new setting, depending on how small the step-size parameter is when the change happens.

Yet another limitation of the proposed algorithms is that the step-size parameter they compute at each time step is universal, meaning that a single step-size parameter is computed that is used to update the parameter vector in all directions. Remember that in principle, stochastic gradient descent is a component-wise process in which the step-size parameter for each component can be set separately. The Step-size Ratchet algorithm should ideally follow this principle, and as a result, it will be applicable to more flexible function approximators such as artificial neural networks.

Chapter 9

Closing

In this dissertation, we studied practical online off-policy learning. The objective was to understand the strengths and weaknesses of existing algorithms and also to improve the algorithms where weaknesses are observed. We took a few steps towards this objective. First, we conducted two empirical studies of prominent prediction learning algorithms. Our results provide new insights into how these algorithms compare to each other, their relative merits, and their inter-relationships. We put forth three non-trivial algorithmic ideas and showed that they are useful towards more practical online off-policy learning.

The first main contribution of this dissertation is a novel off-policy prediction learning method: the TDRC algorithm. With the introduction of TDRC, we took a step in closing the gap between the sample efficiency of Gradient-TD algorithms and Off-policy TD(λ).

The second contribution of this dissertation is a new algorithm for off-policy control learning. The new algorithm is called QRC, and is the control variant of the TDRC algorithm. We showed, potentially for the first time, that the QRC algorithm can have practical advantages to simpler unsound algorithms such as Q-learning.

The third and fourth main contributions are in-depth empirical studies of online off-policy prediction learning algorithms on three tasks of increasing complexity. The data showed that no one algorithm solves all tasks better than others. Nevertheless, the empirical studies shed light on the strengths and weaknesses of the algorithms. For the first time, we showed that Emphatic

TD(λ) tends to have a lower error level than other algorithms, but learns more slowly in situations where there are large differences between the target and behavior policies. Within the Gradient-TD family, Proximal GTD2(λ) is probably the slowest to learn, but converges to a lower asymptotic error level than other Gradient-TD algorithms. We learned that TDRC(λ) is probably the most practical choice among all Gradient-TD algorithms studied in this dissertation for solving the problems we considered here. TDRC(λ) provides a standard way of setting the second step-size parameter and in general performs as well as other Gradient-TD algorithms. We learned that algorithms that adapt λ during learning can be inferior to other algorithms when applied to simple problems such as the Collision task; however, they seem to be the most robust across problems.

The fifth main contribution of this dissertation are the Step-size Ratchet and Soft Step-size Ratchet algorithms for adapting Emphatic TD(λ)'s step-size parameter. We showed that both algorithms can be quite effective in speeding up Emphatic TD's learning in high variance environment such as Rooms and High Variance Rooms tasks.

9.1 Future Research Directions

Off-policy learning has come far in the past decades but that fact alone says little about how much farther it has to go if it is to realize its true potential. With every new step that we take towards solving or better understanding a problem, comes new questions that need to be answered, and interesting future directions ripe for exploration. We discuss a few of these below.

Improving Emphatic TD(λ)'s learning speed Emphatic TD(λ) seems to have a lower asymptotic error than other algorithms; however, it is prone to the high variance induced by the product of the importance sampling ratios. A few approaches might help speed up Emphatic-TD's learning that were not pursued in this dissertation. We discuss a few ideas below.

One approach might be through meaningfully defining the interest func-

tion. All studies so far in the literature, have used an interest equal to one in all states for simplicity. The fact that Emphatic TD is affected by the variance more than other algorithms might be due to the increase in the emphasis variable: at each time step, the emphasis variable is incremented by the value returned by the interest function in that state. Setting the interest smaller wherever possible will slow down the rate of increase in the emphasis magnitude and might in turn help control the variance and speed up learning.

We might be able to improve the learning speed of the Emphatic TD(λ, β) algorithm by using an adaptive β parameter. On one hand, the β parameter should be close to γ, to provide low asymptotic error, and on the other hand, it should at times be downsized to allow fast learning. Note that when β becomes smaller, Emphatic TD(λ, β) is expected to perform more similarly to Off-policy TD(λ), and based on the data presented in this dissertation, it might learn faster (note that at $\lambda = 0$, Emphatic TD(λ, β) reduces to Off-policy TD(λ)). In states where large importance sampling ratios are observed, β can be set small, whereas in sates where the importance sampling ratio is not large, β can be set closer to γ, in which case the algorithm reduces to Emphatic TD(λ) and lower asymptotic error can be expected.

Another simple approach to reducing the variance is by clipping the traces in Emphatic TD(λ). Both the eligibility and the followon traces can be cut off if they become larger than a certain value. This approach is in general similar to what has been done in Retrace, ABQ, and Tree Backup algorithms, which try to indirectly control the magnitude of the trace by using a small λ when the importance sampling ratio is large. These algorithms, however, do not directly truncate the eligibility traces. The approach we are suggesting here is to control the magnitude of the traces directly and truncate them if they become larger than a certain value. For example, if the ℓ_2 norm of the eligibility trace or the followon trace is larger than 200, simply set it to 200. In principle, truncating traces should be applicable to any TD method, including Emphatic-TD algorithms. One benefit of truncating the trace is that it provides an intuition for setting the step-size parameter. For example, if the eligibility trace is capped at 200, this is an indication that the step-size should

probably be set smaller than 1/200. A similar approach for truncating traces in TD learning is suggested by Yu, Mahmood, and Sutton (2018).

Investigating Emphatic TD(λ) in the on-policy setting Emphatic TD(λ) was originally proposed to solve the stability problem of off-policy learning. However, Emphatic TD(λ) is different from TD(λ) even in the on-policy case. Previous work showed that Emphatic TD(λ) outperforms TD(λ) when solving the prediction variant of the Mountain car problem (Ghiassian, Rafiee, & Sutton, 2017). Other previous work showed that Emphatic TD(λ) solves counterexamples to on-policy TD(λ) (Gu, Ghiassian, & Sutton, 2019). Gu, Ghiassian, and Sutton (2019) applied Emphatic TD(λ) to three counterexamples that were proposed to show on-policy TD(λ)'s divergence or poor performance on specific problems. Emphatic TD(λ) solved all three counterexamples successfully.

The behavior and performance of Emphatic TD(λ) in the on-policy setting remains largely unclear. Does Emphatic TD(λ) converge to a lower asymptotic error than on-policy TD(λ)? Why can Emphatic TD(λ) solve counterexamples to on-policy TD(λ)? Is Emphatic TD(λ) in general a better algorithm than on-policy TD(λ)? Should all temporal-difference learning use emphasis?

Investigating Emphatic TD's fixed point and its relationship to the $\overline{\text{VE}}$ Emphatic TD's fixed point is the minimum of the $\overline{\text{PBE}}$ weighted by the emphatic weightings. TD's fixed point is the minimum of the $\overline{\text{PBE}}$ weighted by the behavior policy distribution. Minimum of the $\overline{\text{VE}}$ in the function approximation case is not necessarily the same as any of the minimums found by the Emphatic or non Emphatic algorithms discussed in this dissertation. We observed in the experiments that Emphatic TD(λ) tends to have a lower asymptotic error than other algorithms, which means that in the problems that we considered, the distance between the minimum of the $\overline{\text{PBE}}$ weighted by the emphatic weightings and the minimum of the $\overline{\text{VE}}$ was smaller than the distance between the minimum of the $\overline{\text{PBE}}$ weighted by the behavior policy distribution and the minimum of the $\overline{\text{VE}}$. An interesting future research

direction is to try to understand the conditions under which Emphatic TD converges to a lower $\overline{\text{VE}}$ than other algorithms.

Reducing variance in off-policy prediction learning through adapting the bootstrapping parameter The strategy for controlling the variance by adapting λ based on the magnitude of importance sampling ratios (like Tree Backup, V-trace, and ABTD do), has proven limited on simple problems such as the Collision task. Tree backup, V-trace, and ABTD adapt the bootstrapping parameter of Off-policy TD(λ), and are not consistent in the statistical sense. An alternative idea for adapting λ is to set it based on how certain the agent is in the value of the state it is bootstrapping from. When the agent is certain about the value of the next state, λ can be set to 0 to reduce variance. As certainty decreases, λ can be set to larger values.

Investigating the asymptotic error level of Proximal GTD2(λ) In the Collision and the Rooms experiments, we observed that the Proximal GTD2(λ) algorithm converged to a lower error level than other Gradient-TD algorithms. An interesting future research direction is to investigate why this happened in these specific problems and if this is something more general and might happen in other problems as well. One reason might be that on those problems, Proximal GTD2 converged to a minimum of the $\overline{\text{PBE}}$ that was different from (and had a smaller $\overline{\text{VE}}$ than) the minimum that the other algorithms converged to. Note that the $\overline{\text{PBE}}$ can have more than one minimum in the function approximation case.

Improving the TDRC algorithm An important next step is to better understand the conditions on the regularization parameter β and whether we can truly remove the second step-size parameter η. The current theorem does not remove conditions on η; in fact, it has the same conditions as TDC. We hypothesize that β should make $\mathbf{v}$ converge more quickly, and so removes the need for the step-size parameter for the secondary weights to be larger. Further, the conditions on η and β both depend on domain specific quantities

that are generally difficult to compute.

Another important next step is to thoroughly investigate if the empirical results of TDRC and QRC hold in a broader range of environments and settings. The results in this work suggest that TDRC could potentially be a replacement for the widely used TD algorithms. It is only a small modification to an existing TD implementation, and so would not be difficult to adopt. But, to make such a bold claim, much more evidence is needed, particularly because TD has been shown to be so successful for many years.

Improving the applicability of the Step-size Ratchet Algorithm One important next step is to develop a vectorized version of the Step-size Ratchet algorithm. In principle, stochastic gradient descent is a component-wise process in which the step-size parameter for each component can be set separately; there is no rule that prevents us from setting each step-size element in each direction separately. The Step-size Ratchet algorithm should ideally follow this principle. However, our proposed algorithms do not follow this principle. The next step would be to derive an algorithm that ratchets down each component of the step-size parameter, one for each direction.

Another important next step is to apply the Step-size Ratchet and Soft Step-size Ratchet to other reinforcement learning algorithms, such as GTD(λ) or TDRC(λ). In this dissertation, we only applied Step-size Ratchet and Soft Step-size Ratchet to the Emphatic TD(λ) algorithm because our goal was to speed up the algorithm that had the lowest asymptotic error level on the problems that we studied. It remains unclear how much or if Step-size Ratchet and Soft Step-size Ratchet can benefit other algorithms. Our expectation is that the Step-size Ratchet and Soft Step-size Ratchet algorithms can greatly increase the learning speed of other algorithms especially in high variance environments and in cases that a large value of bootstrapping parameter is used.

More empirical studies A simple but useful future research direction is to conduct additional online off-policy prediction learning experiments on new

tasks. The new tasks could be simulations of real-world tasks like the ones studied in this dissertation, or they can be random MDPs. With more tasks, trends over results can be established. These trends can be used to better understand the advantages and disadvantages of existing algorithms. Hopefully, these empirical results and trends can eventually be used for developing novel and more effective algorithms.

9.2 Closing Thoughts

Finally, we would like to stress the need for thorough empirical studies. A thorough examination is necessary to obtain the understanding that is critical to using algorithms successfully and with confidence. There is a need for thorough empirical studies, but they take time, and a proper presentation of them takes space. While our study is not the last word, it does contribute to the growing database of reliable results comparing modern off-policy learning algorithms.

References

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

Barto, A. G., Sutton, R. S., Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics, 13*(5), pp. 835–846. Reprinted in J. A. Anderson and E. Rosenfeld (1988), *Neurocomputing: Foundations of Research,* pp. 535–549.

Bellemare, M. G., Naddaf, Y., Veness, J., Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research, 47,* pp. 253–279.

Borkar, V. S., Meyn, S.P. (2000). The O.D.E. Method for Convergence of Stochastic Approximation and Reinforcement Learning. *SIAM Journal of Control and Optimization, 38*(2), pp. 447–469.

Boyan, J. A. (1999). Least-squares temporal difference learning. In *Proceedings of the 16th International Conference on Machine Learning,* pp. 49–56.

Boyan, J.A. (2002). Technical Update: Least-Squares Temporal Difference Learning. *Machine Learning, 49*(2), pp. 233–246.

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

Dabney, W., Barto, A. G. (2012). Adaptive step-size for online temporal difference learning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence.*

Dai, B., Albert, S., Lihong, L., Lin, X., Niao, H., Zhen, L., Jianshu, C., Le, S. (2018). SBEED: Convergent reinforcement learning with nonlinear function approximation. In *Proceedings of the 35th International Conference on Machine Learning,* pp. 1125–1134.

Dann, C., Neumann, G., Peters, J. (2014). Policy evaluation with temporal-differences: A survey and comparison. *Journal of Machine Learning Research, 15,* pp. 809–883.

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

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I. and Legg, S. (2018). IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning,* pp. 1407–1416.

Feng, Y., Li, L., Liu, Q. (2019). A kernel loss for solving the bellman equation. In *Proceedings of the 33rd Neural Information Processing Systems Conference,* pp. 15430–15441.

Geist, M., Scherrer, B. (2014). Off-policy learning with eligibility traces: A survey. *Journal of Machine Learning Research 15,* pp. 289–333.

Ghiassian, S., Patterson, A., Garg, S., Gupta, D., White, A., White, M. (2020). Gradient temporal-difference learning with regularized corrections. In *Proceedings of the 37th International Conference on Machine Learning,* pp. 3524–3534.

Ghiassian, S., Patterson, A., White, M., Sutton, R. S., White, A. (2018). Online off-policy prediction. ArXiv:1811.02597.

Ghiassian, S., Rafiee, B., Sutton, R. S. (2016). A first empirical study of emphatic temporal difference learning. In *Workshop on Continual Learning and Deep Learning at the Conference on Neural Information Processing Systems.* Also, ArXiv: 1705.04185.

Ghiassian, S., Sutton, R. S. (2021a). An Empirical Comparison of Off-policy Prediction Learning Algorithms on the Collision Task. ArXiv: 2106.00922.

Ghiassian, S., Sutton, R. S. (2021b). An Empirical Comparison of Off-policy Prediction Learning Algorithms in the Four Rooms Environment. ArXiv:2109.05110.

Glorot, X., Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics,* pp. 249–256.

Gu, X., Ghiassian, S., Sutton, R. S. (2019). Should All Temporal Difference Learning Use Emphasis? ArXiv:1903.00194.

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

Hallak, A., Tamar, A., Munos, R., Mannor, S. (2016). Generalized emphatic temporal-difference learning: Bias-variance analysis. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence,* pp. 1631–1637.

Jacobsen, A., Schlegel, M., Linke, C., Degris, T., White, A., White, M. (2019). Meta-descent for online, continual prediction. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence,* pp. 3943–3950.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. ArXiv: 1611.05397.

Jiang, R., Zahavy, T., Xu, Z., White, A., Hessel, M., Blundell, C., van Hasselt, H. (2021). Emphatic Algorithms for Deep Reinforcement Learning. In *Proceedings of the 38th International Conference on Machine Learning.*

Juditsky, A., Nemirovski, A., Tauvel, C. (2011). Solving variational inequalities with stochastic mirror-prox algorithm. *Stochastic Systems 1*(1), pp. 17–58.

Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. ArXiv:1412.6980.

Liu, B., Liu, J., Ghavamzadeh, M., Mahadevan, S., Petrik, M. (2015). Finite-Sample Analysis of Proximal Gradient TD Algorithms. In *Proceedings of the 31st International Conference on Uncertainty in Artificial Intelligence,* pp. 504–513.

Liu, B., Liu, J., Ghavamzadeh, M., Mahadevan, S., Petrik, M. (2016). Proximal Gradient Temporal-Difference Learning Algorithms. In *Proceedings of the 25th International Conference on Artificial Intelligence,* pp. 4195–4199.

Littman, M. L., Sutton, R. S., Singh, S. (2002). Predictive representations of state. In *Proceedings of 14th Neural Information Processing Systems Conference,* pp. 1555–1561.

Mahadevan, S., Liu, B., Thomas, P., Dabney, W., Giguere, S., Jacek, N., Gemp, I., Liu, J. (2014). Proximal reinforcement learning: A new theory of sequential decision making in primal-dual spaces. ArXiv: 1405.6757.

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

Maei, H. R. (2011). *Gradient temporal-difference learning algorithms.* PhD thesis, University of Alberta.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D. (2015). Human level control through deep reinforcement learning. *Nature 518*(7540), pp. 529–533.

Modayil, J., Sutton, R. S. (2014). Prediction driven behavior: Learning predictions that drive fixed responses. In *AAAI-14 Workshop on Artificial Intelligence and Robotics.*

Moore, A. W. (1990). *Efficient memory-based learning for robot control.* PhD thesis, University of Cambridge.

Munos, R., Stepleton, T., Harutyunyan, A., Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *Proceedings of 29th Neural Information Processing Systems Conference,* pp. 1046–1054.

Nachum, O., Dai, B., Kostrikov, I., Chow, Y., Li, L., Schuurmans, D. (2019a). Algaedice: Policy gradient from arbitrary experience. ArXiv:1912.02074.

Nachum, O., Chow, Y., Dai, B., Li, L. (2019b). Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. ArXiv:1906.04733.

Precup, D., Sutton, R. S., Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. In *Proceedings of the 18th International Conference on Machine Learning*, pp. 417–424.

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

Rafiee, B., Ghiassian, S., White, A., Sutton, R. S. (2019). Prediction in Intelligence: An Empirical Comparison of Off-policy Algorithms on Robots. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems,* pp. 332–340.

Reddi, S. J., Kale, S., Kumar, S. (2019). On the convergence of Adam and beyond.ArXiv:1904.09237.

Ring, M. B. (in preparation). Representing knowledge as forecasts (and state as knowledge).

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

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

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Proceedings of eighth Neural Information Processing Systems Conference,* pp. 1038–1044.

Sutton, R. S., Barto, A. G. (2018). *Reinforcement Learning: An Introduction,* second edition. MIT press.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, Cs., Wiewiora, E. (2009). Fast gradient-descent algorithms for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning,* pp. 993–1000.

Sutton, R. S., Maei, H. R., Szepesvári, C. (2008). A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation. In *Proceedings of 21st Neural Information Processing Systems Conference,* pp. 1609–1616.

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

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems,* pp. 761–768.

Sutton, R. S., Precup, D., Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence, 112,* pp. 181–211.

Sutton, R. S., Whitehead, S. D. (1993). Online learning with random representations. In *Proceedings of the 10th International Conference on Machine Learning,* pp. 314–321.

Tanner, B., Sutton, R. S., (2005). TD(λ) networks: temporal-difference networks with eligibility traces. In *Proceedings of the 22nd international conference on Machine learning,* pp. 888–895.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation, 6*(2), pp. 215–219.

Thomas, P. S. (2015). *Safe reinforcement learning.* PhD thesis, University of Massachusetts Amherst.

Touati, A., Bacon, P. L., Precup, D., Vincent, P. (2018). Convergent tree-backup and retrace with function approximation. ArXiv:1705.09322.

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

van Hasselt, H., Guez, A., Silver, D. (2016). Deep reinforcement learning with double Q-learning. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence.*

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* PhD thesis, University of Cambridge.

Watkins, C. J. C. H., Dayan, P. (1992). Q-learning. *Machine Learning, 8,* pp. 279–292.

White, A. (2015). *Developing a predictive approach to knowledge.* PhD thesis, University of Alberta.

White, M. (2017). Unifying Task Specification in Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning,* pp. 3742–3750.

White, A., White, M. (2016). Investigating practical linear temporal difference learning. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems,* pp. 494–502.

Young, K., Tian, T. (2019). Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. ArXiv:1903.03176.

Yu, H., Mahmood, A. R., Sutton, R. S. (2018). On generalized bellman equations and temporal-difference learning. *The Journal of Machine Learning Research, 19*(1), pp. 1864–1912.

Zhang, R., Dai, B., Li, L., Schuurmans, D. (2019). GenDICE: Generalized Offline Estimation of Stationary Values. In *Proceedings of eighth International Conference on Learning Representations.*

Appendix A

Various Forms of Importance Sampling Placement for Off-policy TD(λ) and a Comparative Study of Them

In the main body of the dissertation, in Chapter 6, we showed that Off-policy TD(λ) update rules can be written in two different forms. We showed that these two forms are equivalent step by step given that the eligibility trace vector is set to zero at the beginning of each run. The role of this chapter is to show that another set of update rules used for Off-policy TD(λ) in the literature, is equal to the two other forms discussed in the main body of the dissertation in expectation.

A.1 The Third Form of Importance Sampling Placement for Off-policy TD(λ)

Here, we show that the update rules for Off-policy TD(λ) can be written in a third form, which is not necessarily equivalent to the two other forms step by step, but is equivalent to the two other forms in expectation. The third form of the update rules are:

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} \rho_t(R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1}) - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_{t-1}\gamma_t\lambda_t\mathbf{z}_{t-1} + \mathbf{x}_t \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t.
\end{aligned}$$

The second and third update rules are the same as the one we discussed in Chapter 6. We only need to show that the TD-error defined here, $\delta_t \stackrel{\text{def}}{=} \rho_t(R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1}) - \mathbf{w}_t^\top \mathbf{x}_t$, is equal in expectation to the TD-error defined previously $\delta_t \stackrel{\text{def}}{=} \rho_t(R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t)$. We only need to show that the last terms in the two TD-errors are equal in expectation. This is because the first two terms of the TD-error are exactly the same in the two updates, and:

$$\begin{aligned}
&= \mathbb{E}_b\Big[\rho_t \left(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t\right) \mid S_t\Big] \\
&= \mathbb{E}_b\Big[\rho_t \left(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1}\right) \mid S_t\Big] - \mathbb{E}_b\Big[\rho_t \left(\mathbf{w}_t^\top \mathbf{x}_t\right) \mid S_t\Big]. \qquad \text{(A.1)}
\end{aligned}$$

We now only need to show that:

$$\mathbb{E}_b\Big[\mathbf{w}_t^\top \mathbf{x}_t \mid S_t\Big] = \mathbb{E}_b\Big[\rho_t \left(\mathbf{w}_t^\top \mathbf{x}_t\right) \mid S_t\Big].$$

This is straightforward because $\mathbb{E}_b\Big[\mathbf{w}_t^\top \mathbf{x}_t \mid S_t\Big] = \mathbf{w}_t^\top \mathbf{x}_t$ and:

$$\mathbb{E}_b\Big[\rho_t \left(\mathbf{w}_t^\top \mathbf{x}_t\right) \mid S_t\Big] = \mathbf{w}_t^\top \mathbf{x}_t \underbrace{\mathbb{E}_b[\rho_t \mid S_t]}_{=1} = \mathbf{w}_t^\top \mathbf{x}_t.$$

A.2 An Empirical Study of Different Importance Sampling Placements

We just showed that correcting or not correcting the last term in the TD-error, $\mathbf{w}_t^\top \mathbf{x}_t$, does not change the expected update. All algorithms that we discussed throughout the dissertation use the TD-error, and for all of them, the last term in the TD-error can be corrected or it can be left with no correction. Which approach should we choose?

Here, we conduct a simple comparison of all algorithms we discussed in Chapter 3, with and without correcting the last term of the TD-error. The experimental setting remains the same as the original Collision task experiment, with the same number of time steps and same algorithm instances applied to the task.

We applied all algorithms to the Collision task with and without correcting the $\mathbf{w}_t^\top \mathbf{x}_t$ term and summarized the results in Figure A.1 and Figure A.2. Let

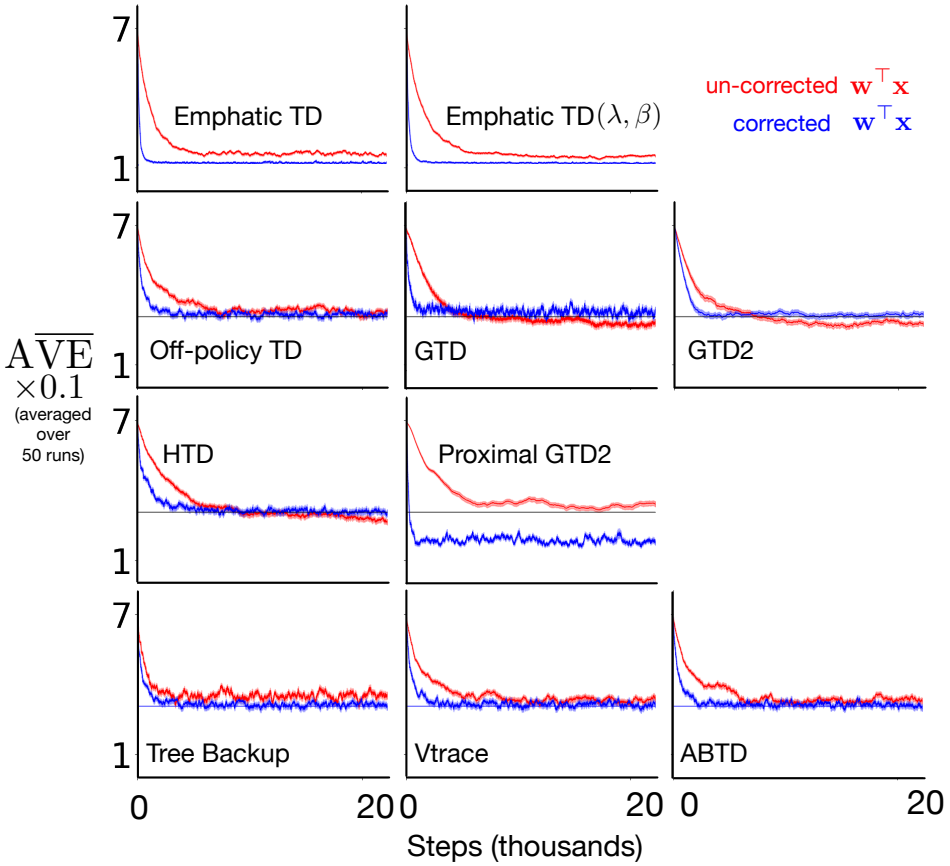


Figure A.1: Different ρ placements for the **Collision** problem when $\lambda = 0$. The curves are optimized for the area under the curve. The figures compare how ρ placement can affect the performance of each method. Blue is when the whole TD-error term is corrected and red is when $v(S_t)$ is not corrected.

us first focus on the learning curves in Figure A.1. The learning curves are average errors over 50 runs, and the ones that are chosen have the minimum area under the learning curve for each algorithm. In all cases, when the whole TD-error was corrected, the algorithm learned faster. The difference between the learning speed when $\mathbf{w}_t^\top\mathbf{x}_t$ was corrected and when it was not corrected was statistically significant.

We also plotted the area under the learning curve for all algorithms with corrected and un-corrected $\mathbf{w}_t^\top\mathbf{x}_t$ in Figure A.2. For all algorithms, when the whole TD-error was corrected, the parameter sensitivity curve was wider, meaning that it was easier to choose a good step-size for the algorithm.

The conclusion that we make from this experiment is simple: it is best if

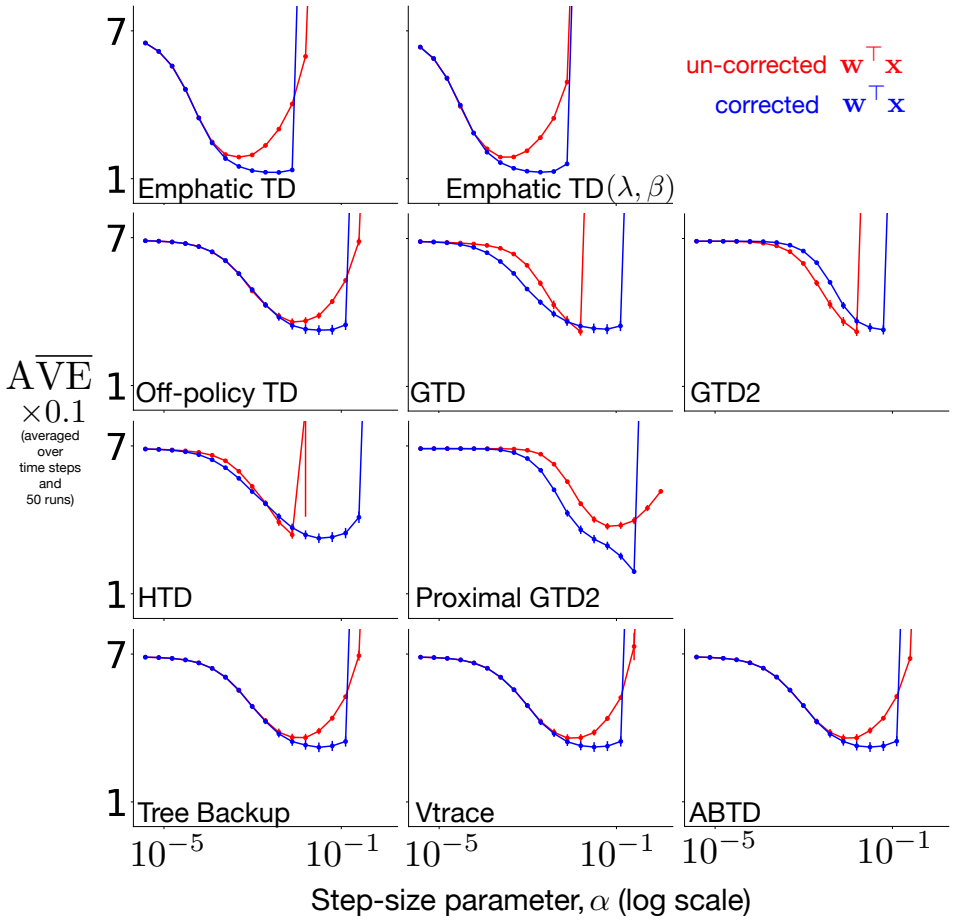


Figure A.2: **Collision** problem when $\lambda = 0$. The curves are optimized for the area under the curve. The figures compare how ρ placement can affect the performance of each method. Blue is when the whole TD-error term is corrected and red is when $v(S_t)$ is not corrected.

the whole TD-error including the $\mathbf{w}_t^\top \mathbf{x}_t$ term is corrected with the importance sampling ratio.

Appendix B

A Summary of Prediction Learning Algorithms and All Update Rules

Here, we first briefly introduce the eleven algorithms used in our empirical studies. As the reader might remember, the eleven algorithms are intended to include all the best candidate algorithms for off-policy prediction learning with linear function approximation. We will then provide the update rules for all algorithms as a single point of reference that can be used for implementation.

B.1 A Summary of Algorithms

In this section, we briefly introduce the eleven algorithms used in our empirical study. These eleven are intended to include all the best candidate algorithms for off-policy prediction learning with linear function approximation.

Off-policy TD(λ) (Precup, Sutton, & Dasgupta, 2001) is the off-policy variant of the original TD(λ) algorithm (Sutton, 1988) that uses importance sampling to reweight the returns and account for the differences between the behavior and target policies. This algorithm has just one set of weights and one step size parameter.

Our study includes five algorithms from the Gradient-TD family. *GTD(λ)* and *GTD2(λ)* are based on algorithmic ideas introduced by Sutton et al., (2009), then extended to eligibility traces by Maei (2011). *Proximal GTD2(λ)* (Mahadevan et al., 2014; Liu et al., 2015; Liu et al., 2016) is a "mirror descent"

version of GTD2 using a saddle-point objective function. These algorithms approximate stochastic gradient descent (SGD) on an alternative objective function, the mean squared projected Bellman error. *HTD(λ)* (Hackman, 2012; White & White, 2016) is a "hybrid" of GTD(λ) and TD(λ) which becomes equivalent to classic TD(λ) where the behavior policy coincides with the target policy. *TDRC(λ)* is a promising recent variant of GTD(λ) that adds regularization. All these methods involve an additional set of learned weights (beyond that used in $\hat{v}$) and a second step-size parameter, which can complicate their use in practice. TDRC(λ) offers a standard way of setting the second step-size parameter, which makes this less of an issue. All of these methods are guaranteed to converge with an appropriate setting of their two step-size parameters.

Our study includes two algorithms from the Emphatic-TD family. Emphatic-TD algorithms attain stability by up- or down-weighting the updates made on each time step by Off-policy TD(λ). If this variation in the emphasis of updates is done in just the right way, stability can be guaranteed with a single set of weights and a single step-size parameter. The original emphatic algorithm, *Emphatic TD(λ)*, was introduced by Sutton, Mahmood, and White (2016). The variant *Emphatic TD(λ, β)*, introduced by Hallak et al., (2016), has an additional parameter, $\beta \in [0, 1]$, intended to reduce variance.

The final three algorithms in our study can be viewed as different attempts to address the problem of large variations in the product of importance sampling ratios. If this product might become large, then the step-size parameter must be set small to ensure there is no overshoot—and then learning may be slow. All these methods attempt to control the importance sampling product by changing the bootstrapping parameter from step to step (Yu, Mahmood, & Sutton, 2018). Munos et al., (2016) proposed simply putting a cap on the importance sampling ratio at each time step; they explored the theory and practical consequences of this modification in a control context with their Retrace algorithm. *V-trace(λ)* (Espeholt et al., 2018) is a modification of Retrace to make it suitable for prediction rather than control. Mahmood et al., (2017) developed a more flexible algorithm that achieves a similar effect. Their algo-

rithm was also developed for control; to apply the idea to prediction learning we had to develop a nominally new algorithm, *ABTD(ζ)*, that naturally extends ABQ(ζ) from control to prediction. A full development of ABTD(ζ) can be found in Chapter 6 of the dissertation. Finally, *Tree Backup(λ)* (Precup, Sutton, & Singh, 2000) reduces the effective λ by the probability of the action taken on each time step. Each of these algorithms (or their control predecessors) have been shown to be very effective on specific problems.

B.2 Update Rules

Here, we provide the update rule for all off-policy prediction learning algorithms we used in this dissertation.

Off-policy TD(λ):

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_t(\gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}_t) \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t
\end{aligned}$$

GTD(λ):

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_t(\gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}_t) \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{v}_{t+1} &\leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}} \Big[\delta_t \mathbf{z}_t - (\mathbf{v}_t^\top \mathbf{x}_t)\mathbf{x}_t \Big] \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t - \alpha \gamma_{t+1}(1 - \lambda_{t+1})(\mathbf{v}_t^\top \mathbf{z}_t)\mathbf{x}_{t+1}
\end{aligned}$$

TDRC(λ):

$$
\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_t(\gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}_t) \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{v}_{t+1} &\leftarrow \mathbf{v}_t + \alpha\Big[\delta_t \mathbf{z}_t - (\mathbf{v}_t^\top \mathbf{x}_t)\mathbf{x}_t\Big] - \alpha \mathbf{v}_t \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha\delta_t \mathbf{z}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\mathbf{v}_t^\top \mathbf{z}_t)\mathbf{x}_{t+1}
\end{aligned}
$$

GTD2(λ):

$$
\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_t(\gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}_t) \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{v}_{t+1} &\leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}}\Big[\delta_t \mathbf{z}_t - (\mathbf{v}_t^\top \mathbf{x}_t)\mathbf{x}_t\Big] \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha(\mathbf{v}_t^\top \mathbf{x}_t)\mathbf{x}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\mathbf{v}_t^\top \mathbf{z}_t)\mathbf{x}_{t+1}
\end{aligned}
$$

HTD(λ):

$$
\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_t(\gamma \lambda \mathbf{z}_{t-1}^\rho + \mathbf{x}_t) \qquad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{z}_t^b &\leftarrow \gamma\lambda \mathbf{z}_{t-1} + \mathbf{x}_t \qquad \text{with } \mathbf{z}_{-1}^b = \mathbf{0} \\
\mathbf{v}_{t+1} &\leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}}\Big[\delta_t \mathbf{z}_t - (\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})(\mathbf{v}_t^\top \mathbf{z}_t^b)\Big] \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha\Big[\delta_t \mathbf{z}_t + (\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})(\mathbf{z}_t - \mathbf{z}_t^b)^\top \mathbf{v}_t\Big]
\end{aligned}
$$

Proximal GTD2(λ):

$$
\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
\mathbf{z}_t &\leftarrow \rho_t(\gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}_t) \quad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{v}_{t+\frac{1}{2}} &\leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}} \Big[\delta_t \mathbf{z}_t - (\mathbf{v}_t^\top \mathbf{x}_t)\mathbf{x}_t\Big] \\
\mathbf{w}_{t+\frac{1}{2}} &\leftarrow \mathbf{w}_t + \alpha(\mathbf{v}_t^\top \mathbf{x}_t)\mathbf{x}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\mathbf{v}_t^\top \mathbf{z}_t^\rho)\mathbf{x}_{t+1} \\
\delta_{t+\frac{1}{2}} &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_{t+\frac{1}{2}}^\top \mathbf{x}_{t+1} - \mathbf{w}_{t+\frac{1}{2}}^\top \mathbf{x}_t \\
\mathbf{v}_{t+1} &\leftarrow \mathbf{v}_t + \alpha_{\mathbf{v}} \Big[\delta_{t+\frac{1}{2}} \mathbf{z}_t^\rho - (\mathbf{v}_{t+\frac{1}{2}}^\top \mathbf{x}_t)\mathbf{x}_t\Big] \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha(\mathbf{v}_{t+\frac{1}{2}}^\top \mathbf{x}_t)\mathbf{x}_t - \alpha\gamma_{t+1}(1-\lambda_{t+1})(\mathbf{v}_{t+\frac{1}{2}}^\top \mathbf{z}_t^\rho)\mathbf{x}_{t+1}
\end{aligned}
$$

Emphatic TD(λ):

$$
\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
F_t &\leftarrow \rho_{t-1}\gamma_t F_{t-1} + I_t \quad \text{with } F_0 = I_0 \\
M_t &\stackrel{\text{def}}{=} \lambda_t I_t + (1-\lambda_t) F_t \\
\mathbf{z}_t &\leftarrow \rho_t \left(\gamma_t \lambda \mathbf{z}_{t-1} + M_t \mathbf{x}_t\right) \quad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t
\end{aligned}
$$

Emphatic TD(λ, β):

$$
\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t \\
F_t &\leftarrow \rho_{t-1}\beta F_{t-1} + I_t \quad \text{with } F_0 = I_0 \\
M_t &\stackrel{\text{def}}{=} \lambda_t I_t + (1-\lambda_t) F_t \\
\mathbf{z}_t &\leftarrow \rho_t \left(\gamma_t \lambda \mathbf{z}_{t-1} + M_t \mathbf{x}_t\right) \quad \text{with } \mathbf{z}_{-1} = \mathbf{0} \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t
\end{aligned}
$$

Tree Backup(λ) for prediction:

$$\begin{aligned}
\delta_t^\rho &\stackrel{\text{def}}{=} \rho_t\left(R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top\mathbf{x}_{t+1} - \mathbf{w}_t^\top\mathbf{x}_t\right)\\
\mathbf{z}_t &\leftarrow \gamma_t\lambda_t\pi_{t-1}\mathbf{z}_{t-1} + \mathbf{x}_t \quad \text{with } \mathbf{z}_{-1} = \mathbf{0}\\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha\delta_t^\rho\mathbf{z}_t
\end{aligned}$$

V-trace(λ):

$$\begin{aligned}
\delta_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top\mathbf{x}_{t+1} - \mathbf{w}_t^\top\mathbf{x}_t\\
\mathbf{z}_t &\leftarrow \max(\rho_t, 1)\left(\gamma_t\lambda\mathbf{z}_{t-1} + \mathbf{x}_t\right) \quad \text{with } \mathbf{z}_{-1} = \mathbf{0}\\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t
\end{aligned}$$

ABTD(ζ):

$$\begin{aligned}
\delta_t^\rho &\stackrel{\text{def}}{=} \rho_t\left(R_{t+1} + \gamma_{t+1}\mathbf{w}_t^\top\mathbf{x}_{t+1} - \mathbf{w}_t^\top\mathbf{x}_t\right)\\
\mathbf{z}_t &\leftarrow \gamma_t\nu_{t-1}\pi_{t-1}\mathbf{z}_{t-1} + \mathbf{x}_t \quad \text{with } \mathbf{z}_{-1} = \mathbf{0}\\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha\delta_t^\rho\mathbf{z}_t
\end{aligned}$$