Reinforcement Learning for Continuing Problems Using Average Reward

by

Abhishek Naik

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

Doctor of Philosophy

Department of Computing Science
University of Alberta

Abstract

This dissertation develops simple and practical learning algorithms from first principles for long-lived agents. Formally, the algorithms are developed within the reinforcement learning framework for continuing (non-episodic) problems, in which the agent-environment interaction goes on *ad infinitum*, with the goal of maximizing the average reward obtained per step. The average-reward formulation is under-studied in reinforcement learning with several important open problems.

The first contribution of this dissertation involves the development of foundational one-step average-reward learning methods for prediction and control. The central idea involves using the TD error to estimate the average reward, which enables proofs for convergence in both the on- and off-policy tabular settings. Experimental results show that the algorithms' performance is robust to the values of their parameters.

Next, we extend the above one-step prediction algorithm to make *multi-step* updates using eligibility traces, because multi-step methods can be more sample-efficient. Based on the analysis of a related algorithm, we prove convergence in the on-policy setting with linear function approximation. We also show the first convergence proof in the *off-policy* setting for a multi-step tabular average-reward prediction algorithm.

Finally, we show that standard *discounted* algorithms can be significantly improved if their rewards are centered by subtracting out the rewards' empirical average, which could be changing with time in the control problem. We discuss two ways of estimating the average reward that can be used with any standard discounted algorithm and demonstrate the benefits of reward centering with tabular, linear, and non-linear function approximation.

Preface

The main chapters of this dissertation are based on papers that are either published in conference proceedings, presented at workshops, or are under review.

In particular, some contents of Chapters 1 and 3 have appeared in:

- Wan, Y., Naik, A., & Sutton, R. S. (2021a). Learning and Planning in Average-Reward Markov Decision Processes. *International Conference on Machine Learning*.

The three of us jointly developed the algorithms and wrote the manuscript, of which Yi Wan and I were joint first authors. Yi worked out all the convergence proofs in the paper; those are not included in this dissertation. I performed all the experiments and have included them here. We worked on the literature survey together and have both included it in our dissertations. The algorithm derivations and some additional analysis are new to this dissertation and do not appear in the paper.

Some contents of Chapter 4 have appeared in or will appear in:

- Naik, A., & Sutton, R. S. (2022). Multi-Step Average-Reward Prediction via Differential TD(λ). *Conference on Reinforcement Learning and Decision Making*.
- Naik, A., Yu, H., & Sutton, R. S. (2024). Multi-Step Off-Policy Average-Reward Prediction with Eligibility Traces. *In preparation for submission to a journal*.

I developed all the algorithms and performed all the experiments in both manuscripts. I worked out all the proofs in the first manuscript. The second manuscript builds on the first and contains an additional new family of algorithms which I analyzed with Huizhen (Janey) Yu. I primarily wrote both manuscripts with suggestions from Janey and Rich.

Some contents of Chapter 5 appear in:

- Naik, A., Wan, Y., Tomar, M., & Sutton, R. S. (2024). Reward Centering. *Under review.*

Yi, Rich, and I were involved in the initial conception of the idea. I fleshed out the idea in its current form and performed all the experiments. I also analyzed the resulting algorithms; Yi Wan provided the formal theorem statement. I wrote the manuscript with suggestions from Rich and Yi. Manan performed additional experiments that are not part of this dissertation.

While performing the research presented in this dissertation, I also worked on other projects. In the following list, the first three are directly related to my dissertation topic while the other two are not:

- Naik, A., Shariff, R., Yasui, N., Yao, H., & Sutton, R. S. (2019). Discounted Reinforcement Learning Is Not an Optimization Problem. *Optimization Foundations for Reinforcement Learning Workshop at the Conference on Neural Information Processing Systems.* Also *ArXiv:1910.02140.*
- Naik, A., Abbas, Z., White, A., & Sutton, R. S. (2021). Towards Reinforcement Learning in the Continuing Setting. *Never-Ending Reinforcement Learning workshop at the International Conference on Learning Representations.*
- Wan, Y., Naik, A., & Sutton, R. S. (2021b). Average-Reward Learning and Planning with Options. *Advances in Neural Information Processing Systems.*
- Kudashkina, K., Wan, Y., Naik, A., Sutton, R. S. (2021). Planning with Expectation Models for Control. *ArXiv:2104.08543*
- Naik, A., Chang, B., Karatzoglou, A., Mladenov, M., Chen, M., & Chi, E. H. (2023). Investigating Action-Space Generalization in Reinforcement Learning for Recommendation Systems. *Workshop on Decision Making for Recommendation Systems at the World Wide Web conference (oral presentation).*

"What you get by achieving your goals is not as important as what you become by achieving your goals."

-Zig Ziglar

To my family,

for fostering my curiosity.

Acknowledgments

I was excited and scared in equal measure when I learned Rich had agreed to be my Ph.D. advisor. It was surreal to have the opportunity to work with *the* "Sutton" of the Sutton-and-Barto textbook I had perused over the past year. I do not know what my Master's advisor—Balaraman Ravindran—wrote in his recommendation letter to Rich, and I never asked—my imposter syndrome was off the charts. But Rich was kind and generous with his time from day one. He gave me the independence to find my feet and grow, and was always there when I needed a mentor, a peer, a cheerleader, or a friend. Thanks to Rich, the most important thing I have learned in my Ph.D. is to think clearly.

I am also indebted to several people who helped me see the things that would have taken ages to learn by myself. In particular, Yi and Janey patiently answered the hundreds of questions I sent them over the years; discussions with them, as well as with Arsalan, Zaheer, Kris, Khurram, Kenny, Eric, Tian, Dhawal, Shivam, Roshan, Niko, Banafsheh were rich with insights. I deeply appreciate Dale, Marlos, Martha, and Harm for scrutinizing my work and shaping the directions that I investigated. Martha S's writing workshop helped me identify, appreciate, and strive for crisp writing. I am also thankful to Hengshuai for letting me do fundamental research in the industry, and to Bo, Minmin, and Alexandros for letting me explore application-oriented research at Google's scale. A big shout-out to all the students that I had the pleasure of TAing over five years; taking classes, answering questions, and grading assignments helped me truly understand the basics of RL. I am also grateful for the invaluable computing resources provided by the Digital Research Alliance of Canada.

These few years have been especially delightful with all my friends. I will always cherish the never-ending conversations about research, life, love, Oilers, F1, TV, everything, while kicking around a football in the hallway with Zaheer, who became more of a brother than a flatmate. Over biryanis, birthdays, bike rides, and Bollywood, I learned confidence from Zaheer, self-assuredness from Khurram, to laugh at the smallest things from Raksha, and to not take things too seriously from Paritosh and Manan. Dávid gave wings—no, wheels—to the engineer and racer in me. Abbey's warmth and Suyog's insights made all the meet-ups all the merrier. Saturday-night video calls, laughs, and games with Palli, Himmu, and Maalu got me through the pandemic, as did reminiscing the college days with Ishu, Monu, Serdaar, Stack, and Ujjawal. Hockey with Andy, Daniel, Connor, Vlad, and my teams—Try Hards and Aquila HC—made me look forward to the Edmontonian winters; I hope I always live near mountains for snowboarding. With Javier, Vlad, and Bram, I discovered the thrill of mountain biking in our beautiful river valley. MMRG sessions with Roshan, Joseph, Matt, Abbas, Gautham, Varshini, Gábor, Jordan, and the lot were instrumental in expanding my horizons to interdisciplinary perspectives on figuring out how the mind works, and how it *could* work. Thanks also to Esraa, Esra'a, Shibhansh, Haseeb, Prabhat, Abdul, Richie, Daniel P, and all the lab folks for filling the lab life with laughter. Nerding out about KSP with Connor, Adrian, Aidan were among my favorite moments in the lab. Finally, my gratitude to AlbertaSat for fostering and launching my aspirations of continuing AI research in the space industry.

Some days were much darker than others, but I had Malvika by my side, and her smile can light up even the darkest night. She single-handedly made these last few months an order of magnitude easier. Her passion, selflessness, and resilience drive me to be a better person every day. No one in the world understands me more than Akka, who was always around for a quick laugh. With her, Mihir indulged all my Lego obsessions. And talking to Amma-Pappa over breakfast every morning made sure I never lost sight of what is most important. Thank you for everything.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The high-level goal of this dissertation is to develop simple and practical learning algorithms from first principles for long-lived agents. By agents, I mean any kind of artificial autonomous decision-making systems having a physical manifestation or a virtual existence; the key is their actions have measurable consequences and that they learn to make decisions from their experience. The formal framework that I am using for my research is that of reinforcement learning (RL), which formalizes the interaction of decision-making systems and the world with the goal of maximizing a *reward* signal. I want to study agents that make decisions throughout their lifetimes, so my focus is what are called *continuing* problems in RL. In particular, I consider the average-reward formulation, which is an important way to formalize the objective of long-lived agents in RL.

Continuing problems are ones in which the time span of an agent's decisions' consequences can potentially be of the order of the agent's lifetime. We frequently make such decisions in our lives—choosing academics over a career in sports, choosing research over software engineering, choosing one life partner over another. Now imagine a rover exploring the surface of Mars: if in one expedition it hits a big rock and damages one of its sensors, in subsequent excursions the rover has to continue using the damaged sensor. A common theme in the above problems is that the learning system lives with the consequences of its decisions over its lifetime.

The continuing problem setting is distinct from the more commonly studied episodic problem setting, which better models problems in which the span of decisions' consequences are bound within episodic boundaries. The game of chess is a prototypical example of an episodic problem: each game starts afresh; the consequences of moves are restricted to a single game.

Continuing problems are interesting because the learning systems that I envision inhabit the world we live in and make decisions whose consequences can span arbitrary time spans. Many problems around us are continuing in spirit, and if we intend to build intelligent systems that have general learning abilities, we expect them to solve continuing problems in addition to episodic ones.

In the sequential decision-making literature, continuing problems have been commonly modeled using two formulations: the discounted-reward and the average-reward formulation. In the discounted formulation, the agent is expected to learn to make decisions that maximize the discounted sum of future rewards. An exponentially-decaying weight per time step makes the potentially infinite sum of rewards finite. As a result, rewards further in the future following a decision are *discounted* in favor of more immediate rewards. In the average-reward formulation, the objective for the agent is to maximize the *rate* at which reward is obtained over the long term, which translates to a large sum of rewards over time. There is no discounting; the reward at each time step is weighted equally.

Classically, sequential decision-making problems are cast as Markov decision processes (MDPs), which can then be solved exactly using a rich literature of dynamic-programming (DP) techniques. For continuing problems, the average-reward formulation and solution methods are well-established in the DP literature. However, in reinforcement learning, the average-reward formulation is not as prevalent as the discounted-reward formulation. This is likely because the discounted formulation is also applicable in episodic problems, which are the focus of a vast majority of the RL community (and for which we now have several landmark demonstrations of RL!).

The average-reward formulation for RL is under-studied and has a variety of exciting open problems. My goal for this dissertation was *to advance the frontiers of RL research on continuing problems by developing algorithms based on the average-reward formulation.*

In the following section I outline the existing research in average-reward RL and discuss the gaps filled by the contributions of this dissertation.

Contextualizing the Average-Reward Literature

The average-reward RL literature builds on the well-established methods in the DP literature. Methods that use models of MDPs to compute solutions are often called *planning* methods. Howard first studied average-reward MDPs in 1960 and introduced policy iteration. Based on Bellman's (1957) value iteration, White (1963) proposed *relative* value iteration (RVI) for average-reward MDPs. These methods involve substeps whose complexity is of order the number of states or more, and hence are not well-suited for large problems. Jalali and Ferguson (1989, 1990) were among the first to explore more incremental methods, though their algorithms are limited to special-case MDPs and require referencing the value of a special state–action pair. When I started my Ph.D. in late 2018, the best average-reward planning algorithm appeared to be the planning variant of Abounadi et al.'s (2001) RVI Q-learning: an incremental planning algorithm that is guaranteed to converge when applied to a stream of experience generated from a model. However, RVI Q-planning also relies on specifying a function over state–action pairs. In our paper (Wan, Naik, & Sutton, 2021a), we built on these advances to introduce average-reward planning methods that are convergent without a special reference function.[1]

A major focus in the field of RL are methods that do not assume access to a complete model of an underlying MDP. Such *model-free* methods are often called

[1]We focus on *model-free* methods in this dissertation. For an extended discussion on planning methods, please refer to the paper.

learning methods, while those that also learn a model and plan with it are called *combined* methods. It is also convenient to classify the *problems* that we want to solve. Recall that in the average-reward formulation, agents seek to maximize the average reward per step, or the *reward rate*—this is called the *control* problem. As a sub-step, agents also need to evaluate the average reward of a *target policy* and the corresponding average-reward value function, which we call the *differential* value function. This problem is called the *prediction* problem. Agents that use learning and combined methods collect their own data. If the agent behaves according to the target policy to collect data, we say the agent is solving the *on-policy* setting of the prediction or control problem; if the *behavior* policy is different from the target policy, we call it the *off-policy* setting.

The on-policy setting is typically easier than the off-policy setting and hence has solution methods with better capabilities and guarantees than their off-policy variants. For instance, on-policy average-reward prediction algorithms include Average-Cost TD(λ) (Tsitsiklis & Van Roy, 1999), LSTD(λ) (Konda, 2002), and LSPE(λ) (Yu & Bertsekas, 2009), all of which are guaranteed to converge with linear function approximation. In particular, the three algorithms maintain a running average of the observed rewards, which converges to the true average reward of the target policy; further, the convergence of the value estimates is limited by function approximation, but there are strict bounds on the quality of the estimated solution (e.g., Tsitsiklis and Van Roy, 1999).

Off-policy learning is more challenging and convergence results only exist for the tabular case without any function approximation. In this case, a running average of the observed rewards according to the behavior policy does not converge the true average reward of the target policy. Existing methods for off-policy prediction instead try to estimate the ratio of the steady-state distributions induced by the target and the behavior policies to estimate the average reward (e.g., Liu et al., 2018; Tang et al., 2019; Mousavi et al., 2020; Zhang et al., 2020a,b). Notably, they do not estimate

the differential value function. In addition, these methods operate on a batch of data.

The first contribution of this dissertation is a tabular average-reward off-policy prediction method that converges to both the true average reward of the target policy and its differential value function. The key idea involves a rearrangement of the differential Bellman equations (which shall be defined shortly) that yields a form of the average reward that can be estimated using the TD error even in the off-policy setting (without explicitly estimating any steady-state distributions). Called *Differential TD-learning*, this method is fully *online*—it makes updates as soon as new data is available, without any batching.

For on-policy *control*, there are several algorithms with various guarantees under different conditions: asymptotic convergence, sub-linear regret, or probably approximately correct. These include tabular learning algorithms (e.g., Wheeler & Narendra, 1986; Abbasi-Yadkori et al., 2019), tabular combined algorithms (e.g., Kearns & Singh, 2002; Brafman & Tennenholtz, 2002; Auer & Ortner, 2006; Jaksch et al., 2010), and policy-gradient algorithms (e.g., Sutton et al., 1999b; Marbach & Tsitsiklis, 2001; Kakade, 2001a; Konda, 2002).

Off-policy control is also more challenging, with theoretical guarantees existing only in the tabular case. The earliest tabular average-reward off-policy learning control algorithms that we know of were those introduced (without convergence proofs) by Schwartz (1993) and Singh (1994). Bertsekas and Tsitsiklis (1996) and Das et al. (1999) introduced algorithms with function approximation, but also did not provide convergence proofs. The most important prior algorithm is *RVI Q-learning*, by Abounadi, Bertsekas, and Borkar (1998, 2001; more on this algorithm shortly). They also introduced SSP Q-learning, which is limited to MDPs with a special state that is recurrent under all stationary policies, whereas RVI Q-learning is shown to be convergent for more general MDPs. Ren and Krogh (2001) presented a tabular algorithm and proved its convergence, but their algorithm requires knowledge of properties of the MDP which are not in general known. Gosavi (2004) and Yang et al. (2016)

separately introduced two algorithms and claimed to proved their convergence, but their proofs are not correct (see Wan's (2023) Section 3.5 for details about the former and Appendix D of our paper (Wan et al., 2021a) for the latter).

RVI Q-learning (Abounadi et al., 2001) is actually a family of off-policy algorithms that estimates the average reward by referencing the estimated values of specific state–action pairs. Each different *reference function* over state–action pairs creates an instance of the algorithm, for instance, a weighted average of the value estimates of all state–action pairs, or in the simplest case, the estimate of a single state–action pair's value. For best results, the referenced state–action pairs should be frequently visited; otherwise convergence can be unduly slow. However, if the behavior policy is linked to the target policy (as in ϵ-greedy behavior policies), then knowing which state–action pairs will be frequently visited may be akin to knowing a substantial part of the problem's solution. This motivates the search for a general learning algorithm that does not require a reference function.

Our *Differential Q-learning* is an online off-policy control algorithm that does not involve the specification of a reference function. Like Differential TD-learning, it maintains an explicit average-reward estimate (like Schwartz, 1993; Singh, 1994) which is updated using the TD error. This enables convergence proofs by slightly generalizing the theory of RVI Q-learning.

Most of the algorithms we have discussed so far are *one-step* methods that update their estimates using only the data from a single time step. Algorithms that make *multi-step* updates can be more sample efficient (see, e.g., Sutton & Barto's Chapters 7 and 12). Tsitsiklis and Van Roy's (1999) Average-Cost TD(λ)—mentioned earlier—is a multi-step on-policy prediction algorithm that uses eligibility traces and has strong convergence results with linear function approximation. Building on their theory, we show that a multi-step extension of our one-step Differential TD-learning is also convergent under the same conditions with linear function approximation.

In the average-reward literature thus far, there was no convergent multi-step al-

gorithm for the off-policy setting, even in the tabular case. In this dissertation, we propose a family of multi-step prediction algorithms for the off-policy setting. Using Borkar's (2009) analysis, we show this family of algorithms converges in the off-policy setting to the true reward rate of the target policy and its differential value function. These are the first multi-step average-reward algorithms that are proved to converge in the off-policy setting.

Mathematically, the average-reward formulation is closely related to the discounted-reward formulation. As far back as in 1962, Blackwell pointed out the relation between the discounted value function of a policy, its average reward, and its differential value function. Based on this relation, Tsitsiklis and Van Roy (2002) noted that as the discount factor approaches one, the corresponding discounted value function contains the exact same information as the average reward and the differential value function, combined. Kakade (2001b) also noted that if the discount factor γ is large enough, an optimal policy corresponding to that discount factor (also called a γ-optimal policy) may also obtain the most average reward. In fact, for finite MDPs, an implication of Blackwell's work is that there exists a critical discount factor γ^* such that $\forall \gamma \in [\gamma^*, 1)$, the γ-optimal policy also maximizes the average reward (see Grand-Clément & Petrik's (2023) Theorem 4.6 and Puterman's (1994) Theorem 10.1.4). The critical discount factor is different for each problem, though, and may be arbitrarily large.

In theory, a discounted-reward algorithm can learn the average-reward-optimal policy with a discount factor that is large enough (in the tabular case). However, the variance of methods like Q-learning grows unbounded with a polynomial power of $1/(1-\gamma)$ (Devraj & Meyn, 2021; Qu & Wierman, 2020; Wainwright, 2019; Even-Dar et al., 2003). RL practitioners are all too familiar with the considerable instability issues that tend to occur when using these algorithms with large discount factors.

Devraj and Meyn (2021) noted from Blackwell's decomposition of the discounted value function that the large constant $(r(\pi)/(1-\gamma)$, where $r(\pi)$ denotes the average

reward of the target policy π) in each of the standard discounted values is inconsequential to, say, picking the argmax action. So they suggested that estimating the standard discounted values is unnecessary and proposed that it is enough to estimate *relative* discounted values. They postulated that their *Relative Q-learning* algorithm has bounded variance (that does not depend on γ).

We explore the extreme version of this idea of learning the relative discounted values instead of the standard discounted values: if the rewards are *mean-centered* (that is, have mean zero), there is no constant in the discounted values that scales with $1/(1-\gamma)$. We propose two ways in which the average reward can be estimated to mean-center the rewards. We show that using these reward-centering techniques significantly improves the performance of standard discounted methods such as TD-learning and Q-learning in continuing problems. In particular, their performance does not degrade even with discount factors tending to one.[2]

For more comprehensive and excellent surveys of the average-reward literature, please refer to Wan (2023) or Dewanto et al. (2020); Mahadevan's (1996) review is also a timeless resource.

Layout of the Dissertation

This dissertation has six chapters (including this one). In Chapter 2, I introduce most of the notation and the particular ideas from the average-reward formulation that are important to understand the contributions of this dissertation.

In Chapter 3, I discuss our first set of contributions: tabular one-step algorithms for prediction and control that are convergent in both the on- and off-policy settings without a reference function. I show the theory behind the derivations of Differential TD-learning and Differential Q-learning and compare their updates to existing work. The same chapter contains experiments that show the performance of both algorithms

[2]When $\gamma = 1$, TD-learning (or Q-learning) with reward centering is equivalent to our average-reward Differential TD-learning (or Differential Q-learning) algorithm.

is quite robust to their parameters.

In Chapter 4, I present our second set of contributions. At first, I propose a multi-step variant of one-step Differential TD-learning and show its convergence proof in the on-policy setting with linear function approximation. Next, I show that a simple extension of this new multi-step algorithm to the off-policy setting may not converge even in the tabular case. Finally, I propose a family of multi-step algorithms and show their convergence proofs the tabular off-policy setting. The chapter also contains experiments that validate the theoretical results and intuitions.

In Chapter 5, I show that standard discounted algorithms can perform significantly better if they center their rewards by subtracting out the rewards' empirical average. I propose two straightforward ways of reward centering that can be used with any existing discounted algorithm. Through a series of experiments, I demonstrate the benefits of reward centering with tabular, linear, and non-linear function approximation.

At the ends of Chapters 3, 4, 5, I discuss follow-up research directions specific to each topic. Finally, in Chapter 6, I summarize the contributions and outline my recommendations for future work.

Chapter 2

The Average-Reward Formulation

This chapter briefly introduces ideas and notations of the average-reward formulation that are relevant to this dissertation. In particular, I will present definitions of the average reward and the corresponding value function, and show the corresponding Bellman equations. I will also provide examples to illustrate nuances that are specific to the average-reward formulation, such as *unichain* Markov chains and *communicating* Markov decision processes.

To define the concepts more precisely, we will need the notation of Markov decision processes. The agent-environment interaction within a continuing sequential decision-making problem can be formulated as a finite Markov decision process (MDP): $\mathcal{M} \doteq (\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{R}$ is a set of rewards, and $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the dynamics of the environment. At each of a sequence of discrete time steps $t = 0, 1, 2, \ldots$, the agent receives an indication of a state of the MDP, $S_t \in \mathcal{S}$, and selects, using a Markov behavior policy $b : \mathcal{A} \times \mathcal{S} \to [0, 1]$, an action, $A_t \in \mathcal{A}$, then receives from the environment a reward, $R_{t+1} \in \mathcal{R}$, and the next state, $S_{t+1} \in \mathcal{S}$, and so on. The transition dynamics are such that $p(s', r \mid s, a) \doteq \Pr(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a)$ for all $s, s' \in \mathcal{S}, a \in \mathcal{A}$, and $r \in \mathcal{R}$. The key feature of the continuing problem is that the agent-environment interaction goes on forever. We consider policies in the set of stationary Markov policies throughout this dissertation, and denote this set by Π.

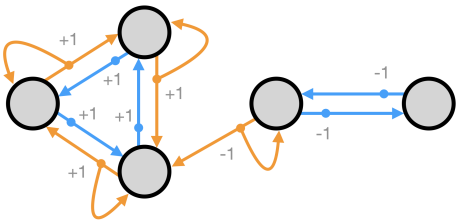


Figure 2.1: Example showing the average reward of a policy can depend on the starting state. Consider a policy that takes the blue action in all states. There are two resulting reward rates: 1 if the agent starts in the left loop; -1 if the agent starts in the right loop.

In general, the average reward per step—or the *reward rate*—of a policy can depend on the start state. Consider the MDP in Figure 2.1 and a policy π_{blue} that deterministically takes the blue action in every state. If the agent starts in one of the states on the left and follows π_{blue} thereafter, it will observe a reward of $+1$ per step; if it starts in one of the states on the right, -1. Mathematically, the reward rate of a policy π can be defined for every state s as:

$$r(\pi, s) \doteq \lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} \mathbb{E}[R_t \mid S_0 = s, A_{0:t-1} \sim \pi]. \tag{2.1}$$

It is convenient to rule out the possibility of the reward rate of the target policy depending on the start state. In particular, we assume that under the target policy there is only one possible limiting distribution for the resulting Markov chain, $\mathbf{d}_\pi : \mathcal{S} \to [0, 1]$, independent of the start state. This is known as the Markov chain being *unichain*. Let the set of policies that induce a unichain be denoted by Π_u.

Under the unichain assumption,

$$r(\pi) \doteq \lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} \mathbb{E}[R_t \mid A_{0:t-1} \sim \pi], \tag{2.2}$$

$$= \sum_s d_\pi(s) \sum_a \pi(a|s) \sum_r p(r \mid s, a)\, r. \tag{2.3}$$

The average-reward state-value function $v_\pi : \mathcal{S} \to \mathbb{R}$ for a policy $\pi \in \Pi_u$ is:

$$v_\pi(s) \doteq \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \sum_{t=1}^{k} \mathbb{E}\left[R_t - r(\pi) \mid S_0 = s, A_{0:t-1} \sim \pi\right], \quad \forall s \in \mathcal{S}. \tag{2.4}$$

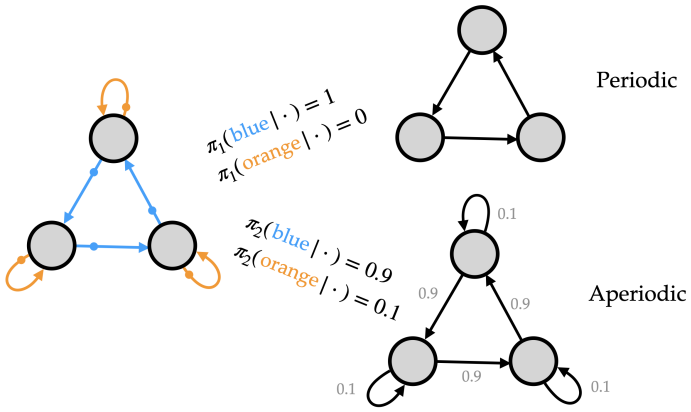


Figure 2.2: Examples of periodic and aperiodic Markov chains. For the 3-state MDP on the left, the first policy induces a periodic Markov chain, while the second policy induces an aperiodic Markov chain.

There is a simpler form of the value function when the Markov chain induced by the policy π in the MDP is *aperiodic*. A Markov chain is *periodic* if there is at least one state s for which the probability of returning to it, $\Pr(S_k = s|S_0 = s)$, is non-zero only for multiples of $k \geq 2$ (Figure 2.2 has an example with $k = 3$). Otherwise, the Markov chain is aperiodic, and the value function then is:

$$v_\pi(s) \doteq \sum_{t=1}^{\infty} \mathbb{E}\big[R_t - r(\pi) \mid S_0 = s, A_{0:\infty} \sim \pi\big], \quad \forall s \in \mathcal{S}. \tag{2.5}$$

Since the average-reward value function involves differences of the rewards with the reward rate, it is often referred to as the *differential* value function.

For the aperiodic case we can define a *differential* return:

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \ldots,$$
$$\text{so that,} \quad v_\pi(s) \doteq \mathbb{E}[G_t \mid S_t = s, A_{t:\infty} \sim \pi], \quad \forall s \in \mathcal{S}.$$

Analogous to the discounted formulation, the differential state-value function and the reward rate satisfies a recursive Bellman equation:

$$v(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big(r - \bar{r} + v(s')\big), \forall\, s \in \mathcal{S}, \tag{2.6}$$

where $v : \mathcal{S} \to \mathbb{R}$ and $\bar{r} \in \mathbb{R}$ are free choices. The Bellman equation can also be

compactly represented in vector-matrix notation:

$$\mathbf{v} = \mathbf{r}_\pi - \bar{r}\,\mathbf{1} + \mathbf{P}_\pi \mathbf{v}, \tag{2.7}$$

where $\mathbf{r}_\pi$ is the expected one-step reward from each state under policy π, $\mathbf{P}_\pi$ is the state-to-state transition matrix under policy π, $\mathbf{1}$ is a vector of all ones, and the vector $\mathbf{v}$ and scalar $\bar{r}$ are free variables. The vector notation can express ideas concisely and hence will be used throughout the dissertation. Unless specified otherwise, lower-case bold letters signify vectors and upper-case bold letters signify matrices.

The Bellman equations for the differential value function specify an under-determined system: for any solution $\big(\mathbf{v}, r(\pi)\big)$, there are infinitely many solutions of the form $\big(\mathbf{v} + c\mathbf{1}, r(\pi)\big)$ for $c \in \mathbb{R}$. Note that $\mathbf{v}$ denotes a general solution to the differential Bellman equation of which $\mathbf{v}_\pi$—the vector form of (2.5)—is a specific solution.

Intuitively, the average-reward formulation separates the infinite stream of rewards into two components: a long-term and a short-term component. The long-term *rate of reward* is captured by the average-reward term; the short-term *sum of rewards in excess of average reward* is captured by the aptly named differential value function. Together, these two quantities summarize the infinite stream of rewards.

Lemma 2.1. *The average of the differential value function weighted by the steady-state distribution is zero, that is,* $\mathbf{d}_\pi^\top \mathbf{v}_\pi = 0$.

Proof. The proof is a simpler version of the proof of Wan et al.'s (2021a) Lemma B.11.

First, note a property of the steady-state distribution $\mathbf{d}_\pi$:

$$\sum_s d_\pi(s) p_\pi(s'|s) = d_\pi(s'), \quad \text{or,} \quad \mathbf{d}_\pi^\top \mathbf{P}_\pi = \mathbf{d}_\pi^\top. \tag{2.8}$$

As a result, $\mathbf{d}_\pi^\top \mathbf{P}_\pi^n = \mathbf{d}_\pi^\top$ for all non-negative integers. Now, re-write the definition of the value function (2.5) in vector form[1]:

$$\mathbf{v}_\pi \doteq \sum_{t=1}^{\infty} \big[\mathbf{P}_\pi^{t-1} \mathbf{r}_\pi - r(\pi)\mathbf{1}\big]. \tag{2.9}$$

[1]The same proof applies to the more general periodic case.

Multiplying $\mathbf{d}_\pi^\top$ on both sides:

$$\begin{aligned}
\mathbf{d}_\pi^\top \mathbf{v}_\pi &= \mathbf{d}_\pi^\top \sum_{t=1}^{\infty} \big[\mathbf{P}_\pi^{t-1}\mathbf{r}_\pi - r(\pi)\mathbf{1}\big] \\
&= \sum_{t=1}^{\infty} \big[\mathbf{d}_\pi^\top \mathbf{P}_\pi^{t-1}\mathbf{r}_\pi - r(\pi)\mathbf{d}_\pi^\top \mathbf{1}\big] \\
&= \sum_{t=1}^{\infty} \big[\mathbf{d}_\pi^\top \mathbf{r}_\pi - r(\pi)\big] \\
&= \sum_{t=1}^{\infty} \big[r(\pi) - r(\pi)\big] \\
&= 0
\end{aligned}$$

□

This property makes intuitive sense: the differential value of a state is the sum of rewards obtained in excess of the average reward, starting from the given state. The differential values are positive in some states and negative in others—on average, zero.

Lemma 2.2 (Based on Tsitsiklis and Van Roy's (1999) Lemma 3)**.** *All the solutions* $(\mathbf{v}, \bar{r})$ *of the differential Bellman equations:* $\mathbf{v} = \mathbf{r}_\pi - \bar{r}\,\mathbf{1} + \mathbf{P}_\pi \mathbf{v}$ *are of the form* $\big(\mathbf{v}_\pi + c\mathbf{1}, r(\pi)\big)$.

Proof. See Tsitsiklis and Van Roy's (1999) proof for their Lemma 3. □

The differential *action-value* function is defined for unichain and aperiodic Markov chains as:

$$q_\pi(s,a) \doteq \sum_{t=1}^{\infty} \mathbb{E}\big[R_t - r(\pi) \mid S_0 = s, A_{1:\infty} \sim \pi\big], \quad \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.10}$$

All the properties of the differential state-value function also apply to the differential action-value function.

In a finite Markov chain, if the probability of a state occurring infinitely often is one, such states are called *recurrent* states; otherwise, *transient*.

So far we have considered the average reward or the reward rate corresponding a particular policy. Let us now consider the *best* reward rate for an MDP. For an unconstrained MDP, the best reward rate depends on the start state. For example, the MDP may have two disjoint sets of states with no policy that passes from one to the other; in this case there are effectively two MDPs, with unrelated rates of reward. A learning algorithm would have no difficulty with such cases—it would optimize for whichever sub-MDP it found itself in—but it is complex to state formally what is meant by an optimal policy. To remove this complexity, it is commonplace to rule out such cases by assuming that the MDP is *communicating*, which just means that there are no states from which it is impossible to get back to the others. As an example, the MDP in Figure 2.1 is not communicating.

Under the communicating assumption, there exists a unique optimal reward rate r^* that does not depend on the start state: $r^* = r(\pi^*, s), \forall s$, where π^* denotes a policy corresponding to the best reward rate. There can be multiple optimal policies. The value function corresponding to such a policy also satisfies a recursive Bellman equation. This Bellman *optimality* equation corresponding to the differential action-value function is:

$$q(s,a) = \sum_{s',r} p(s', r \mid s, a)\big[r - \bar{r} + \max_{a'} q(s', a')\big], \quad \forall s, a. \tag{2.11}$$

where $\mathbf{q}$, $\bar{r}$ are free variables. As with the Bellman evaluation equation (2.6), there are infinite solutions of the Bellman optimality equation of the form $(\mathbf{q}_{\pi^*} + c\mathbf{1}, r^*)$ for $c \in \mathbb{R}$.

In this dissertation we discuss several algorithms to estimate the differential (state- or action-)value function. The problems we are interested in are assumed to have an underlying MDP with a potentially large but finite set of states and actions. The algorithms do not have direct access to the underlying transition or reward dynamics, only the stream of agent-environment interactions. If the observation stream includes the state of the MDP, the agent can approximate the value of each state separately—in

a table. This case is referred to as the *tabular* case. More often, the number of unique states may be too many, and/or the observation stream may only include a lossy encoding of states. In this case, the approximate value function may be represented as a parameterized functional form instead of a table—aptly called the *function approximation* case.

With this background, we can begin understanding the contributions in this dissertation.

Chapter 3

One-step Differential Methods

This chapter establishes the foundational algorithms for the average-reward formulation: one-step tabular model-free methods (a) for the prediction and control problems that are (b) guaranteed to converge in both the on- and off-policy settings.

Tabular one-step methods are the foundational solution methods for any problem formulation. The foundations of average-reward solution methods were laid just over two decades ago. The methods proposed in this chapter strengthen the foundations by improving the generality of existing average-reward methods. In particular,

1. Our *Differential TD-learning* is the first tabular one-step off-policy prediction algorithm that is proved to converge to the average reward and the differential value function of the target policy,

2. Our *Differential Q-learning* is the first general tabular one-step control algorithm that is proved to converge in the off-policy setting without any reference states.

The methods use the TD error to estimate the average reward—a central idea across the chapters of this dissertation.

Much of the contents of this chapter have appeared in a paper that my collaborator—Yi Wan—and I published as joint first authors with Rich Sutton (Wan, Naik, & Sutton, 2021a). Yi worked out all the convergence proofs and included those in his

dissertation. I performed all the experiments, which I have included here along with the algorithm derivations and some analysis that does not appear in the paper.

3.1 Prediction: Theory

In the prediction problem, the goal is to estimate the differential value function and reward rate of a *target* policy while behaving according to a *behavior* policy.

Let us first consider how to estimate the reward rate $r(\pi)$ from the agent's stream of experience: $S_0, A_0, R_1, S_1, \ldots S_t, A_t, R_{t+1}, S_{t+1}, \ldots$.

The most obvious approach is to average all the rewards seen so far. That is, at time step t, after observing rewards $R_1, R_2, \ldots, R_t$, the estimate of the reward rate, $\bar{R}_t$, can be:

$$\bar{R}_t = \frac{1}{t} \sum_{k=1}^{t} R_k. \tag{3.1}$$

Indeed, this *sample average* approach is guaranteed to converge to the true reward rate. To see why, consider the definition of the reward rate (2.3) when we expand the expectation term:

$$r(\pi) = \sum_s d_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\, r,$$

where $d_\pi(\cdot)$ denotes the limiting distribution over states. If the actions are taken according to the policy π, then the law of large numbers guarantees that $\bar{R}_t$ converges to $r(\pi)$.

An incremental version of the sample-average approach (3.1) is the following update at every time step $t \geq 0$:

$$\bar{R}_{t+1} \doteq \bar{R}_t + \beta_t (R_{t+1} - \bar{R}_t), \tag{3.2}$$

where $\beta_t = 1/(t+1)$. More generally, (3.2) converges to $r(\pi)$ as long as the step sizes $\{\beta_t\}$ follow the standard Robbins-Monro conditions:

$$\sum_t \beta_t = \infty, \qquad \sum_t \beta_t^2 < \infty. \tag{3.3}$$

A general way to estimate an expectation using samples

Say we want to estimate $\bar{x} = \mathbb{E}[X]$ from samples of $X : X_1, X_2, \dots X_n$. Then the estimator $\bar{X}_t$ updated at each time step t as:

$$\begin{aligned}\bar{X}_{t+1} &\doteq (1-\alpha_t)\bar{X}_t + \alpha_t X_t \\ &= \bar{X}_t + \alpha_t(X_t - \bar{X}_t)\end{aligned}$$

is guaranteed to converge to $\bar{x}$ if the step sizes follow the Robbins-Monro conditions.

Intuitively, the general rule is:

$$new\_estimate = old\_estimate + step\_size(new\_target - old\_estimate) \quad (3.4)$$

An example of this general rule in use is the TD-learning update. The expectation form of the discounted value function is given by the Bellman equation:

$$v_\pi^\gamma(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\big(r + \gamma v(s')\big), \quad \forall\, s \in \mathcal{S}.$$

The update for the value estimates V after a sample $(S_t, A_t, R_{t+1}, S_{t+1})$:

$$\underbrace{V_{t+1}^\gamma(S_t)}_{\text{new estimate}} \doteq \underbrace{V_t^\gamma(S_t)}_{\text{old estimate}} + \underbrace{\alpha_t}_{\text{step size}} \big(\underbrace{R_{t+1} + \gamma V_t^\gamma(S_{t+1})}_{\text{new target}} - \underbrace{V_t^\gamma(S_t)}_{\text{old estimate}} \big).$$

Assuming the Robbins-Monro conditions for the step sizes, Sutton (1988a) showed the convergence of this sample-based update to the true discounted value function $v_\pi^\gamma(\cdot)$ for all the states that are recurrent under the policy π.

Given a sample-based update for the reward-rate estimate, how should we update the value estimates? Let us consider the differential Bellman equation (2.6) again:

$$v(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\big(r - \bar{r} + v(s')\big), \quad \forall\, s \in \mathcal{S},$$

corresponding to the unichain assumption from Chapter 2 (unless otherwise specified, the assumptions in Chapter 2 are used throughout this dissertation). To estimate this expectation, we can use the general rule outlined in the previous box to get a sample-based update for the estimates V:

$$\begin{aligned} V_{t+1}(S_t) &\doteq V_t(S_t) + \alpha_t\big(R_{t+1} - \bar{R}_t + V_t(S_{t+1}) - V_t(S_t)\big) \\ &\doteq V_t(S_t) + \alpha_t \delta_t. \end{aligned} \tag{3.5}$$

Combined with the update for the reward-rate estimate, we get a complete algorithm for estimating the different value function and the reward rate. Tsitsiklis and Van Roy (1999) proved the convergence for an algorithm very similar to this one under mild technical conditions. Following their naming convention, we call this the *Average Cost TD-learning* algorithm.

Average Cost TD-learning

At time step t, with the knowledge of $(S_t, A_t, R_{t+1}, S_{t+1})$, update the value and reward-rate estimates as:

$$\begin{aligned} V_{t+1}(S_t) &\doteq V_t(S_t) + \alpha_t \delta_t, \\ \bar{R}_{t+1} &\doteq \bar{R}_t + \beta_t (R_{t+1} - \bar{R}_t), \end{aligned}$$

where, $\delta_t \doteq R_{t+1} - \bar{R}_t + V_t(S_{t+1}) - V_t(S_t)$.

However, this algorithm is restricted to the on-policy case. Multiplying the importance-sampling (IS) ratio in both the updates is not enough to extend this algorithm to the off-policy case. The reason is that the IS ratio only corrects the mismatch in the action distribution of the target and behavior policies *given a state*, but does not correct the state distribution. Let us see this mathematically.

Consider the reward-rate update (3.2), now with an IS ratio, $\rho_t = \dfrac{\pi(A_t|S_t)}{b(A_t|S_t)}$,

$$\bar{R}_{t+1} \doteq \bar{R}_t + \beta_t \rho_t (R_{t+1} - \bar{R}_t). \tag{3.6}$$

In expectation, after taking actions according to the behavior policy b for a long time, the reward-rate estimate $\bar{R}_t$ would converge to (the following box explains why):

$$\begin{aligned}\bar{R}_\infty &= \sum_s d_b(s) \sum_a b(a|s) \sum_{s',r} p(s', r|s, a) \left[\frac{\pi(a|s)}{b(a|s)} r\right] \\ &= \sum_s d_b(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\, r \\ &\neq \sum_s d_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\, r = r(\pi).\end{aligned}$$

Hence, the reward-rate update (3.2) restricts Average-Cost TD to the on-policy case, even in the tabular setting.

Equivalent placements of the importance-sampling ratio

Suppose we want to estimate the mean of a random variable X. The probability distribution for X is p, however, we only observe values of X from a distribution q. Then we can use the importance-sampling (IS) ratio to estimate the mean $\bar{x}$ of X:

$$\begin{aligned}\bar{x} &\doteq \sum_x p(X = x)\, x = \mathbb{E}_{x\sim p}[X] \\ &= \sum_x q(X = x) \frac{p(X = x)}{q(X = x)}\, x \\ &= \sum_x q(X = x) \rho_x x = \mathbb{E}_{x\sim q}[\rho_X X],\end{aligned}$$

where, $\rho_x = p(X = x)/q(X = x)$. Now suppose the samples of X arrive in a stream, one at each time step t: X_t. Then the following two update rules for estimates $\bar{X}_t$ both lead to an unbiased incremental estimate of $\bar{x}$ if the step sizes following the Robbins-Monro conditions:

$$\bar{X}_t \doteq \bar{X}_{t-1} + \alpha_t \rho_t (X_t - \bar{X}_{t-1}), \tag{3.7}$$

$$\bar{X}_t \doteq \bar{X}_{t-1} + \alpha_t (\rho_t X_t - \bar{X}_{t-1}), \tag{3.8}$$

where ρ_t is the IS ratio corresponding to X_t. The reason is that given the

update rules and a stream of data $(X_0, X_1, X_2, \ldots, X_{t-1})$,

$$\begin{aligned}
\mathbb{E}_q[\rho_t(X_t - \bar{X}_{t-1})] &= \mathbb{E}_q[\rho_t X_t - \rho_t \bar{X}_{t-1}] \\
&= \mathbb{E}_q[\rho_t X_t] - \mathbb{E}_q[\rho_t \bar{X}_{t-1}] \\
&= \mathbb{E}_q[\rho_t X_t] - \bar{X}_{t-1}\mathbb{E}_q[\rho_t] \\
&= \mathbb{E}_q[\rho_t X_t] - \bar{X}_{t-1} \\
&= \mathbb{E}_q[\rho_t X_t - \bar{X}_{t-1}].
\end{aligned}$$

Hence, we can interchangeably use (3.7) or (3.8).

A different reward-rate update extends Average-Cost TD to the tabular off-policy setting, and is the essence of the 'Differential' family of algorithms:

$$\bar{R}_{t+1} \doteq \bar{R}_t + \beta_t \rho_t \big(R_{t+1} - \bar{R}_t + V_t(S_{t+1}) - V_t(S_t)\big).$$

Let us now derive why this update leads to an unbiased estimate of the target policy's reward rate. The reason is that this update originates from a form of the reward rate that does not involve the state distribution. Recall that the previous reward-rate update did not extend to the off-policy because the IS ratio does not correct the mismatch in the state distribution of the target and behavior policies.

We get a new form of the reward rate by rearranging the Bellman equation:

$$\begin{aligned}
v(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\big(r - \bar{r} + v(s')\big), \quad \forall\, s \in \mathcal{S} \\
v(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\big(r + v(s')\big) - \bar{r} \\
\bar{r} + v(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\big(r + v(s')\big) \\
\bar{r} &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\big(r + v(s')\big) - v(s) \\
\bar{r} &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\big(r + v(s') - v(s)\big) \qquad (3.9)
\end{aligned}$$

This form of the reward rate *does not involve the probability distribution over states.*

Before considering the sample-based off-policy update corresponding to this expectation (3.9), let us first examine the on-policy version. Using the general rule of estimating an expectation via samples, we get:

$$\begin{aligned}\bar{R}_{t+1} &\doteq \bar{R}_t + \beta_t\big(\underbrace{R_{t+1} + V_t(S_{t+1}) - V_t(S_t)}_{\text{new target}} - \bar{R}_t\big) \\ &= \bar{R}_t + \beta_t \delta_t. \end{aligned} \tag{3.10}$$

We have obtained an update for the reward-rate estimate that involves the TD error, just like the update for the value estimates (3.5)!

The off-policy variant is similar:

$$\bar{R}_{t+1} \doteq \bar{R}_t + \beta_t\big(\underbrace{\rho_t\big[R_{t+1} + V_t(S_{t+1}) - V_t(S_t)\big]}_{\text{new target}} - \bar{R}_t\big).$$

From the previous box, we know that multiplying an additional importance-sampling term ρ_t to the rightmost term results in an equivalent update:

$$\begin{aligned}\bar{R}_{t+1} &\doteq \bar{R}_t + \beta_t\big(\rho_t\big[R_{t+1} + V_t(S_{t+1}) - V_t(S_t) - \bar{R}_t\big]\big) \\ &= \bar{R}_t + \beta_t \rho_t \delta_t. \end{aligned} \tag{3.11}$$

Combined with the off-policy updates to the value estimates, we get a tabular average-reward prediction algorithm that is applicable in both the on- and off-policy settings. We refer to this algorithm as *Differential TD-learning*.

Differential TD-learning

At time step t, with the knowledge of $(S_t, A_t, R_{t+1}, S_{t+1})$, update the value and reward-rate estimates as:

$$V_{t+1}(S_t) \doteq V_t(S_t) + \alpha_t \rho_t \delta_t, \tag{3.12}$$

$$\bar{R}_{t+1} \doteq \bar{R}_t + \underbrace{\eta\alpha_t}_{\beta_t} \rho_t \delta_t, \tag{3.13}$$

where, $\delta_t \doteq R_{t+1} - \bar{R}_t + V_t(S_{t+1}) - V_t(S_t)$.

My collaborator, Yi Wan, proved the convergence of tabular Differential TD under mild technical conditions. I state an informal but complete version of the theorem here. The formal theorem statement and its proof are in our paper (Wan, Naik, & Sutton, 2021a) and Section 3.8.2 of Yi's dissertation (Wan, 2023).

Theorem 3.1 (Informal)**.** *If 1) the Markov chain induced by the target policy π is unichain, 2) every state–action pair for which $\pi(a|s) > 0$ occurs an infinite number of times under the behavior policy, 3) the step sizes, specific to each state, are decreased appropriately, and 4) the ratio of the update frequency of the most-updated state to the update frequency of the least-updated state is finite, then the Differential TD-learning algorithm (3.12–3.13) converges, almost surely, $\bar{R}_t$ to $r(\pi)$ and V_t to a solution of v of the Bellman equations (2.6).*

Also note that we replace β_t with $\eta\alpha_t$ without loss of generality for $\eta > 0$. This makes notation simpler for the mathematical analysis. In addition, relating the step sizes for the reward-rate and value estimates in this manner with a scale factor can make it more intuitive for practitioners to set the value of η (e.g., one-tenth, half, or double the step size of the value estimates).

3.2 Prediction: Experiments

In this section I empirically test Differential TD in both the on- and off-policy settings, with Average-Cost TD as a baseline.

The goal of the prediction problem is to estimate the target value function as accurately as possible. One way to quantify the difference is through the mean squared value error of the value estimates $\hat{\mathbf{v}}$ w.r.t. the value function of the target policy $\mathbf{v}_\pi$: $\text{MSVE}(\hat{\mathbf{v}}) = \sum_s d_\pi(s)\big[v_\pi(s) - \hat{v}(s)\big]^2 = \|\mathbf{v}_\pi - \hat{\mathbf{v}}\|^2_{d_\pi}$, where $\mathbf{d}_\pi$ is a distribution over states induced by the target policy π.

In the average-reward setting, MSVE by itself is not enough. Obviously, we have to evaluate the estimated reward rate as well, which we can via a reward-rate error:

$\mathrm{RRE}(\bar{R}) = \big(r(\pi) - \bar{R}\big)^2$. But more importantly, the MSVE requires a modification to be meaningful. Recall that there are infinitely many solutions to the differential Bellman equations (2.6) of the form $\big(\mathbf{v}_\pi + c\mathbf{1}, r(\pi)\big)$ for $c \in \mathbb{R}$. However, MSVE penalizes an arbitrary solution by a factor of c^2, which is undesirable. Tsitsiklis and Van Roy (1999) first encountered this issue and resolved it by comparing the estimate values with the closest solution: $\inf_c \|(\mathbf{v}_\pi + c\mathbf{1}) - \hat{\mathbf{v}}\|^2_{d_\pi}$. Thankfully we do not require a costly infimum operation in the tabular case. It is simple to compute the offset c in the value estimates and simply subtract it out from the value estimates before computing the MSVE. In particular, thanks to Lemma 2.1,

$$\begin{aligned}
\mathbf{d}_\pi^\top \hat{\mathbf{v}} &= \mathbf{d}_\pi^\top (\mathbf{v}_\pi + c\mathbf{1}) \\
&= \mathbf{d}_\pi^\top \mathbf{v}_\pi + c\, \mathbf{d}_\pi^\top \mathbf{1} \\
&= 0 + c = c.
\end{aligned}$$

Given the connection to Tsitsiklis and Van Roy's work, I call this metric: MSVE (TVR).

For both the on- and off-policy experiments, I use the Two Loop task shown in the upper right of Figure 3.1, which is a common illustrative problem (cf. Mahadevan, 1996, Naik et al., 2019). It is a simple continuing MDP with only one action in every state except state 0. Action `left` in state 0 gives an immediate reward of +1 and action `right` leads to a delayed reward of +2 after five steps.

The first experiment was on-policy, with the policy π to be evaluated being the one that randomly picks `left` or `right` in state 0 with probability 0.5. The reward rate corresponding to this policy is 0.3.

For both Average-Cost TD and Differential TD, I tested combinations of step-size parameters α and η in $\{0.025, 0.05, 0.1, 0.2, 0.4\}$ and $\{0.125, 0.25, 0.5, 1, 2\}$ respectively. The step size α was decayed by a factor of 0.9995 at each step. The value estimates and the reward-rate estimate for both algorithms were initialized to zero. For each parameter setting, I performed 30 runs of 10,000 steps each.

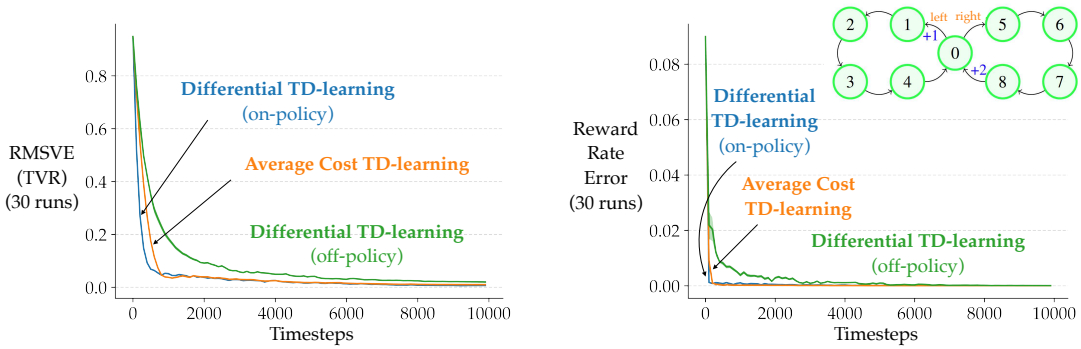


Figure 3.1: Learning curves for Differential TD-learning and Average Cost TD-learning on the Two Loop problem (inset top-right). Exemplary learning curves show all three algorithms tend to zero errors in terms of RMSVE (TVR) and RRE. The standard errors are thinner than width of the solid lines.

The left panel in Figure 3.1 shows the learning curves of the two algorithms (blue and orange) corresponding to the parameters that minimized the *root* MSVE (TVR) averaged over the training period (which reflects the algorithms' rate of learning). The solid line represents the mean and the error bars indicate one standard error (which in many cases was less than the width of the solid lines).

We saw that the RMSVE (TVR) went to zero in a few thousand steps for both on-policy Differential TD-learning and Average Cost TD-learning. The right panel shows the learning curves of the two algorithms (blue and orange) in terms of RRE, which also went to zero for both algorithms.

The plots in Figure 3.2 indicate the sensitivity of the performance of these two algorithms to the two step-size parameters α and η. The average RMSVE (TVR) over all the 10k time steps was equal or lower for Differential TD-learning than Average Cost TD-learning across the range of parameters tested. In addition, on-policy Differential TD-learning was less sensitive to the values of both α and η than Average Cost TD-learning.

For the off-policy experiment I used a behavior policy that picks the `left` and `right` actions with probabilities 0.9 and 0.1 respectively to evaluate the same target policy as before (that picks both the actions with probability 0.5). The parameter

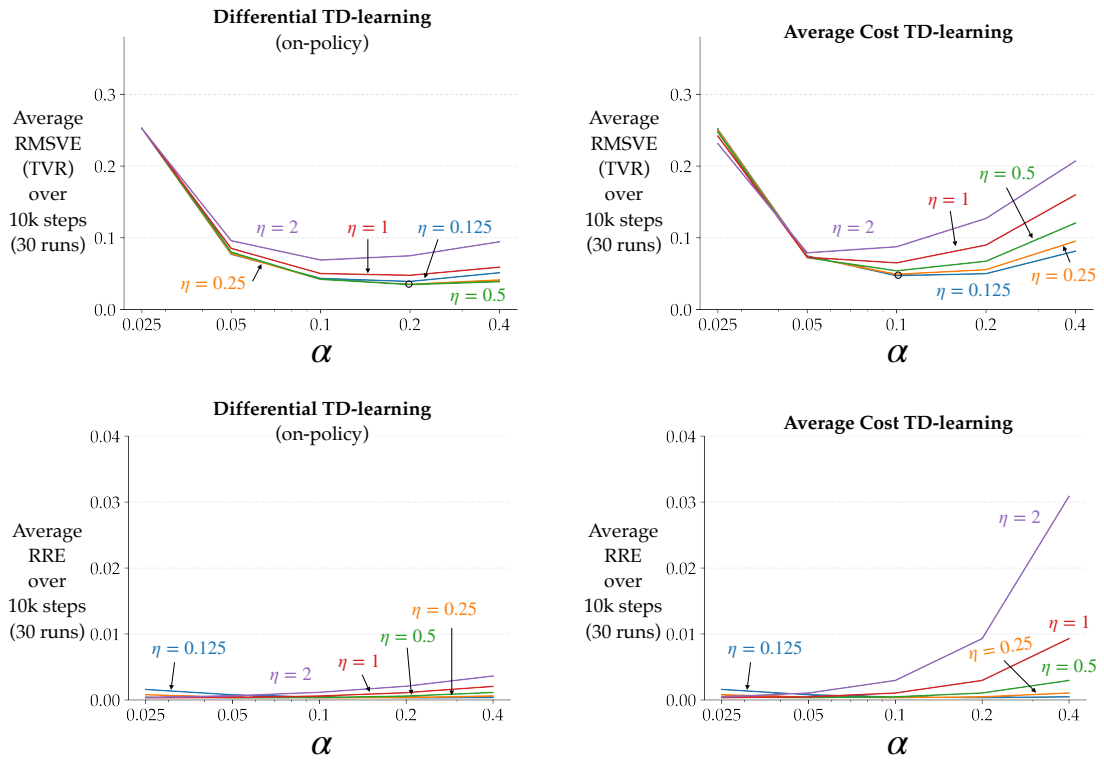


Figure 3.2: Parameter studies for Differential TD-learning and Average Cost TD-learning on the Two Loop problem. The performance of Differential TD-learning in terms of average RMSVE (TVR) *(top)* and average RRE *(bottom)* is less sensitive to the choice of parameters α and η than Average Cost TD-learning. The black circles in the top row denote the parameter configurations for which the RMSVE learning curves are shown in Figure 3.1.

settings and initializations for off-policy Differential TD were the same as the previous on-policy experiment.

The green learning curves in Figure 3.1 correspond to the off-policy Differential TD. Again, both RMSVE (TVR) and RRE went to zero for off-policy Differential TD-learning within a reasonable amount of time. Figure 3.3 shows the sensitivity of both RMSVE (TVR) and RRE of off-policy Differential TD-learning w.r.t. its parameters α and η. Firstly, the off-policy algorithm is more sensitive to its parameters than its on-policy version on this problem. Secondly, the rate of convergence is affected if the parameters are too high or too low, but otherwise it is relatively insensitive to different choices of η. Note that in the tabular setting, the average-reward estimate

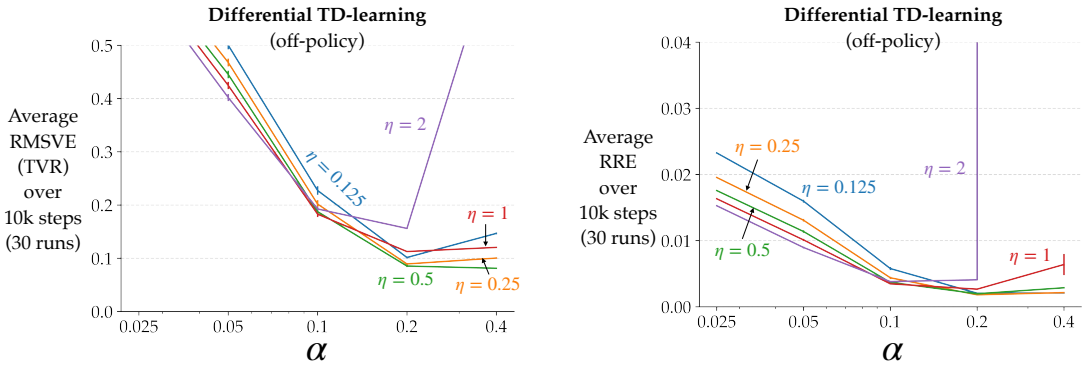


Figure 3.3: Parameter studies for off-policy Differential TD-learning on the Two Loop task. Performance in terms of both metrics is sensitive to the choice of step size α but does not depend much on η.

is updated at every step while the value of a particular state (or state–action pair) is updated relatively less frequently. This might explain why $\eta < 1$ typically leads to better performance in the tabular setting.

These experiments show that on- and off-policy Differential TD-learning algorithms can accurately estimate the value function and the reward rate of a given target policy, as expected from Theorem 3.1. In addition, on-policy Differential TD-learning can be easier to use than Average Cost TD-learning.

Before moving to the control problem, let us take a moment to reflect. Differential TD uses the TD error to update its average-reward estimate because it enables an unbiased estimate of the average reward even in the off-policy setting. However, in the on-policy setting, is there any benefit to using the TD error? Or perhaps a drawback?

Characterizing the difference due to the TD error and the conventional error

The only difference between Differential TD learning and Average-Cost TD learning is how the average-reward estimate is updated:

$$\text{Average-Cost TD:} \quad \bar{R}_{t+1} \doteq \bar{R}_t + \eta\alpha_t\big(R_{t+1} - \bar{R}_t\big),$$

$$\text{Differential TD:} \quad \bar{R}_{t+1} \doteq \bar{R}_t + \eta\alpha_t\big(R_{t+1} - \bar{R}_t + V_t(S_{t+1}) - V_t(S_t)\big).$$

We know that in the on-policy setting, both updates result in an unbiased estimate of the true average reward $r(\pi)$ of the target policy π. The difference would then manifest in the learning process: perhaps one update leads to faster convergence or lower asymptotic variance.

Given that Average-Cost TD only uses the observed rewards to update the average-reward estimate, it is reasonable to hypothesize that it may estimate the average reward faster. Meanwhile, Differential TD uses the values to update the average-reward estimate, and since the values are also being estimated in parallel, their inaccuracies may cause the average-reward estimation to be relatively slower. On the other hand, for a similar reason, we expect Differential TD to have lower asymptotic variance: the per-state expected TD error is zero but the per-state expected error between the one-step reward and the true average reward is not zero.

For intuition, consider the case when both the reward-rate estimate and the value estimates are accurate (learned or otherwise), that is, $\bar{R} = r(\pi)$ and $V(s) = v_\pi(s), \forall s$. The error per time step—for every $(S_t, A_t, R_{t+1}, S_{t+1})$—in the two cases is:

$$\text{Conventional error: } \Delta_t^A \doteq R_{t+1} - r(\pi),$$
$$\text{TD error: } \Delta_t^T \doteq R_{t+1} - r(\pi) + v_\pi(S_{t+1}) - v_\pi(S_t).$$

It is hard to infer anything directly from these per-step errors; the *expected* error per step—given the current state s—is more informative. In case of the conventional error,

$$\begin{aligned}\mathbb{E}[\Delta_t^A \mid S_t = s, A_t \sim \pi] &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\, r - r(\pi) \\ &= r_\pi(s) - r(\pi), \end{aligned} \tag{3.14}$$

where $r_\pi(s)$ denotes the expected one-step reward from state s.[1] The expected per-

[1]Recall that $r(\pi) = \sum_s d_\pi(s)\, r_\pi(s)$ under the unichain assumption.

step conventional error is non-zero. On the other hand, in case of the TD error,

$$\begin{aligned}\mathbb{E}[\Delta_t^T \mid S_t = s, A_t \sim \pi] &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r - r(\pi) + v_\pi(s')\right] - v_\pi(s) \\ &= 0, \end{aligned} \tag{3.15}$$

where the last equality holds due to the differential Bellman equation (2.6). Stated differently, the average TD error is zero per step (unsurprisingly) whereas in general the average conventional error is not.

A natural extension is the hypothesis that the per-step variance of the TD error is lower than that of the conventional error, that is,

$$\mathbb{V}[\Delta_t^T \mid S_t = s, A_t \sim \pi] \leq \mathbb{V}[\Delta_t^A \mid S_t = s, A_t \sim \pi]. \tag{3.16}$$

This property is true for MDPs where the reward and transition dynamics are fully deterministic. When the value estimates are accurate, the per-step TD error is zero (its expectation is equal to its sample value). On the other hand, the per-step conventional error is non-zero in general even with fully deterministic dynamics.

However, I found a counterexample that shows this hypothesis is not true in general. The key idea is if the induced Markov chain is stochastic, the per-step variance of the TD error may be larger than that of the conventional error, despite the per-step expected TD error being zero. I first derive the general expressions for the variance of the two errors and then present the counterexample.

First, I compute the general expression for the variance of the conventional error.

$$\begin{aligned}\mathbb{V}[\Delta_t^A \mid S_t = s, A_t \sim \pi] &= \mathbb{V}_\pi[\Delta_t^A \mid S_t = s] \\ &= \mathbb{V}_\pi[R_{t+1} - r(\pi) \mid S_t = s] \\ &= \mathbb{E}_\pi\Big[\big(R_{t+1} - r(\pi) - \mathbb{E}_\pi[R_{t+1} - r(\pi)|S_t = s]\big)^2 \mid S_t\Big] \\ &= \mathbb{E}_\pi\Big[\big(R_{t+1} - r(\pi) - [r_\pi(s) - r(\pi)]\big)^2 \mid S_t\Big] \quad \text{(from (3.14))} \\ &= \sum_s d_\pi(s) \sum_{s',r} p_\pi(s', r \mid s)\big(r - r_\pi(s)\big)^2, \end{aligned}$$

where, p_π denotes the joint distribution induced by the policy π over the next state and reward given a state. Next, the expression for the variance of the TD error:

$$\begin{aligned}
\mathbb{V}[\Delta_t^T \mid S_t = s, A_t \sim \pi] &= \mathbb{V}_\pi[\Delta_t^T \mid S_t = s] \\
&= \mathbb{V}_\pi[R_{t+1} - r(\pi) + v_\pi(S_{t+1}) - v_\pi(S_t) \mid S_t = s] \\
&= \mathbb{E}_\pi\Big[\big(R_{t+1} - r(\pi) + v_\pi(S_{t+1}) - v_\pi(S_t) \\
&\qquad - \mathbb{E}_\pi[R_{t+1} - r(\pi) + v_\pi(S_{t+1}) - v_\pi(S_t) \mid S_t = s]\big)^2 \mid S_t\Big] \\
&= \mathbb{E}_\pi\Big[\big(R_{t+1} - r(\pi) + v_\pi(S_{t+1}) - v_\pi(S_t)\big)^2 \mid S_t\Big] \quad \text{(from (3.15))} \\
&= \sum_s d_\pi(s) \sum_{s',r} p_\pi(s', r \mid s)\big(r - r(\pi) + v_\pi(s') - v_\pi(s)\big)^2
\end{aligned}$$

Consider the three-state Markov chain with rewards (also called a Markov reward process, or MRP) shown in the adjoining Figure 3.4. The average reward $r(\pi)$ for the policy π that induced this MRP is 0. The steady-state distribution and the differential values of the three states are respectively $\mathbf{d}_\pi = [1/2, 1/4, 1/4]^\top$ and $\mathbf{v}_\pi = [0, -1, 1]^\top$. Then,

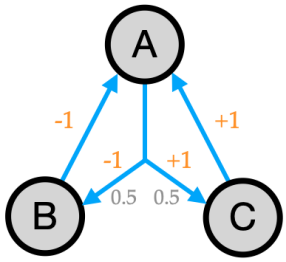


Figure 3.4: The three-state counterexample.

$$\begin{aligned}
\mathbb{V}[\Delta_t^A \mid S_t = s, A_t \sim \pi] &= \frac{1}{2}\Big(\frac{1}{2}(1)^2 + \frac{1}{2}(-1)^2\Big) + \frac{1}{4}0^2 + \frac{1}{4}0^2 \\
&= \frac{1}{2}, \\
\mathbb{V}[\Delta_t^T \mid S_t = s, A_t \sim \pi] &= \frac{1}{2}\Big(\frac{1}{2}\big(-1 - 0 + (-1) - 0\big)^2 + \frac{1}{2}\big(1 - 0 + 1 - 0\big)^2\Big) + \frac{1}{4}0^2 + \frac{1}{4}0^2 \\
&= \frac{1}{2}\Big(\frac{1}{2}4 + \frac{1}{2}4\Big) = 2.
\end{aligned}$$

Thus, on this problem, $\mathbb{V}[\Delta_t^T \mid S_t = s, A_t \sim \pi] > \mathbb{V}[\Delta_t^A \mid S_t = s, A_t \sim \pi]$, which disproves the hypothesis (3.16).

To conclude, in general, the per-step variance of the TD error is not lower than that of the conventional error in the on-policy setting. However, we may expect lower variance if the underlying dynamics of the problem are more deterministic.

3.3 Control: Theory

We now consider the control problem, where the objective is to find the optimal policy. Recall that optimal policies are defined as those corresponding to the reward rate r^*, and that r^* does not depend on the starting state in a communicating MDP. We seek a learning algorithm that achieves r^*.

Having established the essentials in the prediction section, it is easy to see what the control version of the Differential family of algorithms would be. We begin by writing the expectation forms of the *action values* and the reward rate, and use sample-based updates to estimate them (the action values are preferred for control over state values because action values enable faster action selection). We begin by restating the Bellman optimality equation for differential action values:

$$q(s,a) = \sum_{s',r} p(s',r|s,a)\big[r - \bar{r} + \max_{a'} q(s',a')\big], \quad \forall s,a. \tag{3.17}$$

We can rearrange the terms to get an expectation form of the reward rate:

$$\bar{r} = \sum_{s',r} p(s',r|s,a)\big[r + \max_{a'} q(s',a') - q(s,a)\big], \quad \forall s,a. \tag{3.18}$$

Using our general rule to use samples to estimate expectations, we arrive at the *Differential Q-learning* algorithm.

Differential Q-learning (tabular)

At time step t, with the knowledge of $(S_t, A_t, R_{t+1}, S_{t+1})$, update the value and reward-rate estimates as:

$$Q_{t+1}(S_t,A_t) \doteq Q_t(S_t,A_t) + \alpha_t \delta_t, \tag{3.19}$$

$$\bar{R}_{t+1} \doteq \bar{R}_t + \eta\alpha_t\delta_t, \tag{3.20}$$

$$\text{where,} \quad \delta_t \doteq R_{t+1} - \bar{R}_t + \max_{a'} Q_t(S_{t+1},a') - Q_t(S_t,A_t). \tag{3.21}$$

As with Differential TD-learning, my collaborator, Yi Wan, has proved the convergence of Differential Q-learning under mild technical conditions.[2] Again, I present the informal but complete theorem statement here; please refer to our paper (Wan, Naik, & Sutton, 2021a) or Section 3.8.3 of Yi's dissertation (Wan, 2023) for more details.

Theorem 3.2 (Informal)**.** *If 1) the MDP is communicating, 2) the solution of q in (2.11) is unique up to a constant, 3) the step sizes, specific to each state–action pair, are decreased appropriately, 4) all the state–action pairs are updated an infinite number of times, and 5) the ratio of the update frequency of the most-updated state–action pair to the update frequency of the least-updated state–action pair is finite, then the Differential Q-learning algorithm (3.19–3.21) converges, almost surely, $\bar{R}_t$ to r^*, Q_t to a solution of q in 2.11, and $r(\pi_t)$ to r^*, where π_t is any greedy policy w.r.t. Q_t.*

The off-policy control problem is particularly challenging, and theoretical results are available only for the tabular, discrete-state setting without function approximation. The earliest tabular average-reward off-policy learning control algorithms that we know of were those introduced (without convergence proofs) by Schwartz (1993) and Singh (1994). Both of their algorithms are similar to Differential Q-learning with key differences. The former does not update the reward-rate estimate after exploratory actions, citing a skew in the approximation of the true reward rate due to the mismatch in the exploratory and exploitative actions. However, this fear is unfounded. (3.18) is true for all actions, so there is no need to waste useful information obtained via exploratory actions. Singh (1994) noted this and proposed his algorithm to update the reward-rate estimate at each step. However, he added the constraint that the value of a reference state-action pair is always grounded to zero, citing the

[2]Note that there is no importance-sampling ratio ρ_t involved in Differential Q-learning even though it is guaranteed to converge in the off-policy setting. The reason is that the action A_t—coming from the behavior policy—is already taken and hence there is no question about correcting its effect. This is same reason why discounted Q-learning (Watkins & Dayan, 1992) does not have any importance-sampling ratio.

concern that the estimated values might potentially be quite large (in particular, the offset c). This is a fair concern, but this technique may or may not help depending on the choice of the reference state-action pair. Moreover, our work (Wan et al., 2021a) provides a convergence proof for the algorithm and proposes a technique so that there is no offset in the learned estimates (that is, $c = 0$).

The most important prior algorithm is RVI Q-learning, introduced by Abounadi, Bertsekas, and Borkar (1998, 2001). RVI Q-learning is actually a family of off-policy algorithms, a particular member of which is determined by specifying a function that references the estimated values of specific state–action pairs and produces an estimate of the reward rate. The update at time step t is:

$$Q_{t+1}(S_t, A_t) \doteq Q_t(S_t, A_t) + \alpha_t\big(R_{t+1} - f(Q_t) + \max_{a'} Q_t(S_{t+1}, a') - Q_t(S_t, A_t)\big). \tag{3.22}$$

Assuming f—which we call the *reference function*—follows certain conditions, Abounadi et al. guarantee that $f(Q_\infty) = r^*$ and the corresponding argmax policy is an optimal policy. Examples of valid f functions include a weighted average of the value estimates of all state–action pairs, or in the simplest case, the estimate of a single state–action pair's value. For best results, the referenced state–action pairs should occur frequently, otherwise convergence can be unduly slow (illustrated in the next section).

However, as mentioned in Chapter 1, if the behavior policy is linked to the target policy (as in ϵ-greedy behavior policies), then knowing which state–action pairs will be frequently visited may be to know a substantial part of the problem's solution. For example, in learning an optimal path through a maze from diverse starting points, the frequently visited state–action pairs are likely to be those on the shortest paths to the goal state. To know these would be tantamount to knowing a priori the best paths to the goal. This observation motivates the search for a general learning algorithm that does not require a reference function. And that is exactly the void that Differential Q-learning fills.

Interestingly, RVI Q-learning and Differential Q-learning make the same updates to Q_t in special cases. For RVI Q-learning, the special case is when the reference function is the mean of all state–action pairs' values. For Differential Q-learning, the special case is when $\eta = \frac{1}{|\mathcal{S}||\mathcal{A}|}$. These special cases are not particularly good for either algorithm, and therefore their special-case equivalence tells us little about the relationship between the algorithms in practice. In RVI Q-learning, it is generally better for the reference function to emphasize state–action pairs that are frequently visited rather than to weight all state–action pairs equally (see the next section for an example). In Differential Q-learning, the special-case setting of $\eta = \frac{1}{|\mathcal{S}||\mathcal{A}|}$ would often be much too small on problems with large state and action spaces.

3.4 Control: Experiments

In this section I present empirical results for Differential Q-learning. I first compared the performance of Differential Q-learning with that of RVI Q-learning in the tabular setting, in which both algorithms have convergence results. I found that Differential Q-learning performs as well as RVI Q-learning and is more robust to its parameters compared to RVI Q-learning. These trends also hold in the results with linear function approximation (for which there are no convergence results yet).

Tabular experiments

The first experiment uses the Access-Control Queuing task (Sutton & Barto, 2018). This task involves customers queuing up to access to one of 10 servers. The customers have differing priorities (1, 2, 4, or 8), which are also the rewards received if and when their service is complete. At each step, the customer at the head of the queue is either accepted and allocated a free server (if any) or is rejected (in which case a reward of 0 is received). This decision is made based on the priority of the customer and the number of currently free servers, which together constitute the state of this average-reward MDP. The rest of the details of this test problem are exactly as described by

Sutton and Barto (2018: Section 10.3).

I applied tabular Differential Q-learning and RVI Q-learning to this task, each for 30 runs of 80,000 steps, and each for a range of step sizes $\alpha \in \{0.0015625, 0.00625, 0.025, 0.1, 0.4\}$.[3] For Differential Q-learning, η was chosen from $\{0.125, 0.25, 0.5, 1, 2\}$. RVI Q-learning was run with three kinds of reference functions suggested by Abounadi et al. (2001): (1) the value of a single reference state–action pair, for which we considered all possible 88 state–action pairs, (2) the maximum value of the action-value estimates, and (3) the mean of the action-value estimates. Both algorithms used an ϵ-greedy behavior policy with $\epsilon = 0.1$ and no annealing.

A typical learning curve is shown in Figure 3.5. A point on the solid line denotes reward rate over the last 2000 time steps, and the shaded region indicates one standard error. While this learning curve is for Differential Q-learning, the learning curves for both algorithms typically started at around 2.2 and plateaued at around 2.6, with different parameter settings leading to different rates of learning. A reward rate of 2.2 corresponds to a policy that accepts every customer irrespective of their priority or the number of free servers—with positive rewards for every accept action, such a policy is learned rapidly in the first few time steps starting from a zero initialization of value estimates (i.e., a random policy). The optimal performance was close to 2.7.

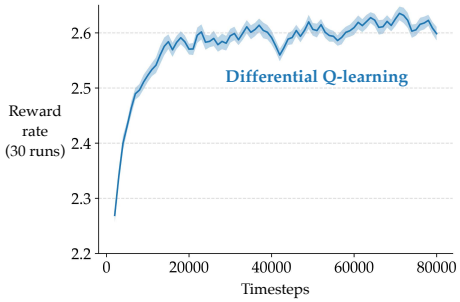


Figure 3.5: A typical learning curve for the Access-Control Queuing task.

Figure 3.6 shows parameter studies for each algorithm. Plotted is the reward rate averaged over all 80,000 steps, reflecting their rates of learning. The error bars

[3]Note that constant step sizes do not follow the Robbins-Monro conditions, and hence the conditions of Theorem 3.2 are violated. However, we explicitly made this choice in the control experiments within this dissertation because this line of work is motivated by the "big-world" perspective in which agents must continually learn and adapt to changes of a vastly complex world (Sutton et al., 2022). Constant step sizes can help in tracking the best solution as opposed to converging to it (Sutton et al., 2007).

indicate one standard error, which at times is less than the width of the solid lines.

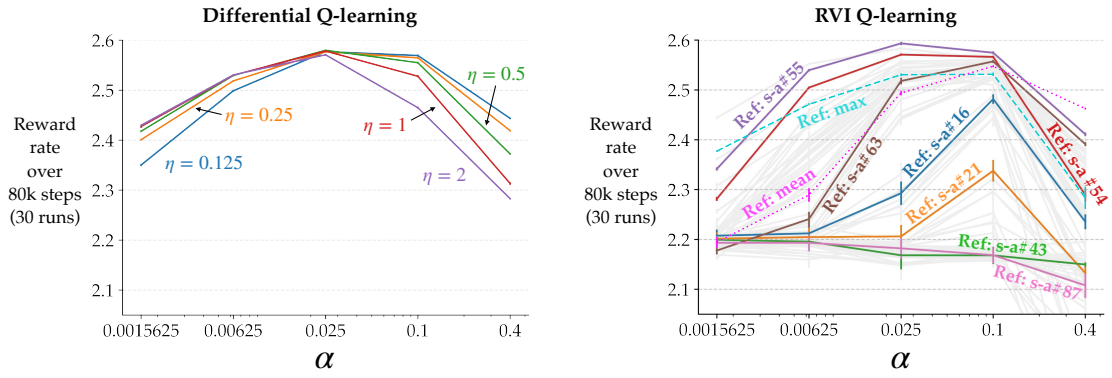


Figure 3.6: Parameter studies showing the sensitivity of the two algorithms' performance to their parameters. *Left:* Differential Q-learning's rate of learning varied little over a broad range of its parameter η. *Right:* RVI Q-learning's rate of learning depended strongly on the choice of the reference function. The solid greyed-out lines mark the performance for each of the 88 state–action pairs considered individually as the single reference pair, with a few representative ones highlighted (labelled as 'Ref: s-a'). The dotted lines correspond to the reference function being the mean or the max of all the action-value estimates.

We saw that Differential Q-learning performed well on this task for a wide range of parameter values (left panel). Its two parameters did not interact strongly; the best value of α was independent of the choice of η. Moreover, the best performance for different η values was roughly the same.

RVI Q-learning also performed well on this task for the best choice of the reference state–action pair, but its performance varied significantly for the various choices of the reference function and state–action pairs (right panel).

A closer look at the data revealed a correlation between the performance of a particular reference state–action pair and how frequently it occurs under an optimal policy. For example, state–action pairs 55 and 54 occurred frequently and also resulted in good performance. They correspond to states when only two servers are free and the customer at the front of the queue has priority 8 and 4 respectively, and the action is to accept. This is the optimal action in this state. In contrast, the performance was poor with state–action pairs 43 and 87, which occurred rarely. They correspond

to states when all 10 servers are free, a condition that rarely occurs in this problem. Finally, the mean of value estimates of all state–action pairs performs moderately well as a reference function. These observations lead us to a conjecture: *an important factor determining the performance of RVI Q-learning with a single reference state–action pair is how often that pair occurs under an optimal policy.* This is problematic because knowing which state–action pairs occur frequently under an optimal policy is tantamount to knowing the solution of the problem we set out to solve.

The conjecture might lead us to think that the reference function that is the max over all action-value estimates would always lead to good performance because the corresponding state–action pair would occur most frequently under an optimal policy, but this is not true in general. For example, consider an MDP with a state that rarely occurs under any policy. Let all rewards in the MDP be zero except a positive reward from that state. Then the highest action value among all state–action pairs is corresponding to this rarely-occurring state.

To conclude, our experiments with the Access-Control Queuing task show that the performance of RVI Q-learning can vary significantly over the range of reference functions and state–action pairs. On the other hand, Differential Q-learning does not use a reference function and can be significantly easier to use.

Linear experiments

The second set of experiments involve the linear variants of Differential Q-learning and RVI Q-learning, that is, in which the action-value function is estimated as a linear combinations of the features of the states and actions. We consider the setting in which at each time step t, the agent observes a feature vector $\mathbf{x}_t \in \mathbb{R}^d$ representing the state of the environment, takes a discrete action A_t, observes the next feature vector $\mathbf{x}_{t+1}$ and the scalar reward signal R_{t+1}. The agent approximates the action-value function at time step t as: $\hat{q}_t \doteq (\mathbf{w}_t^A)^\top \mathbf{x}_t$, where $\mathbf{w}^A$ denotes d-dimensional learnable weights corresponding to the A-th action.

Differential Q-learning (linear)

At time step t, with the knowledge of $(\mathbf{x}_t, A_t, R_{t+1}, \mathbf{x}_{t+1})$, update the reward-rate estimate and the per-action weights as:

$$\mathbf{w}_{t+1}^{A_t} \doteq \mathbf{w}_t^{A_t} + \alpha_t \, \delta_t \, \mathbf{x}_t, \tag{3.23}$$

$$\bar{R}_{t+1} \doteq \bar{R}_t + \eta \, \alpha_t \, \delta_t, \tag{3.24}$$

where, $\delta_t \doteq R_{t+1} - \bar{R}_t + \max_a (\mathbf{w}_t^a)^\top \mathbf{x}_{t+1} - (\mathbf{w}_t^{A_t})^\top \mathbf{x}_t$.

Compared to Differential Q-learning, extensions of the tabular version of RVI Q-learning (Abounadi et al. 2001) to function approximation are not as straightforward, even for linear function approximation. RVI Q-learning requires the value of a reference function f to be computed at every time step t, where f is a function over the current estimates of the value estimates $\hat{q}_t$. Some difficulties that arise with the first attempts of extending the reference functions suggested by Abounadi et al. to the function approximation setting:

- Reference function is the mean of all action-value estimates: $f(\hat{q}_t) \doteq \frac{1}{|\mathcal{S}||\mathcal{A}|} \sum_{s,a} \hat{q}_t(s,a)$
  It is easy to see why the computation of this quantity is problematic in the function approximation setting: unlike the tabular setting, the agent does not have access to the underlying states.[4]
- Reference function is the max of all action-value estimates: $f(\hat{q}_t) \doteq \max_{s,a} \hat{q}_t(s,a)$
  In the tabular setting, it is straightforward to compute the max of the action-value function over all state–action pairs. In the function approximation setting, computing this quantity would again require access to all the underlying states, which the agent does not have.

[4] An alternative problem setting when function approximation is used is when the agent does have access to the underlying states, but they are too many to enumerate (e.g., in a table). In this case $|\mathcal{S}|$ is either unknown, or too large (making $\frac{1}{|\mathcal{S}|}$ too small).

- Reference function is the action-value estimate of a single reference state–action pair (s_0, a_0): $f(\hat{q}_t) \doteq \hat{q}_t(s_0, a_0)$

  Again, the agent does not have access to any underlying state in the function approximation setting. Instead, one might consider using a value of a *reference feature vector* with an action as the reference function. The question then becomes what the reference feature vector should be, among the infinite choices in $\mathbb{R}^d$.

  Based on the observations in the tabular setting, an obvious hypothesis is that the performance of RVI Q-learning with a reference feature vector would depend on the frequency with feature vectors similar to the reference feature vector occur under the optimal policy for the given problem.

Based on the above discussion, we can attempt creating a couple of reference functions for the function approximation setting, for instance, the action-value estimate corresponding to the *first* feature vector the agent observes along the action of moving left. There is no way to compute the max exactly, but perhaps we can try using the maximum of the set of estimated action values corresponding to the feature vectors when they are observed. These are just some first attempts; further research is required to develop theoretically-grounded reference functions for the function approximation setting.

If we have good ways of computing such reference functions at each time step, the linear function approximation version of RVI Q-learning would update the weights similar to linear Differential Q-learning with $\delta_t \doteq R_{t+1} - f(\hat{q}_t) + \max_a (\mathbf{w}_t^a)^\top \mathbf{x}_{t+1} - (\mathbf{w}_t^A)^\top \mathbf{x}_t$ (and no update to a reward-rate estimate).

I performed some preliminary experiments in the linear function approximation setting using the PuckWorld and Catcher domains from the PyGame Learning Environment (Tasfi, 2016).

In the PuckWorld problem, the agent needs to reach the position marked by the

green circle, which moves to a different location after every few time steps. A still of the environment is shown in the top-left panel of Figure 3.7. At every time step, the agent can take one of four actions — left, right, up, down — which move the agent in that direction by a small amount. Repeated actions in the same direction build some velocity in that direction, which decays at an exponential rate at every time step. At every time step, the agent gets a reward proportional to its distance to the goal position. This reward is typically negative and becomes zero when the agent reaches the goal position. At every time step, the agent observes a six-dimensional feature vector of its horizontal position, vertical position, horizontal velocity, vertical velocity, target's horizontal position, target's vertical position. The positions and velocities are scaled to lie in $[0, 1]$ and $[-1, 1]$ respectively. After a regular interval of time steps, the goal position is uniform-randomly initialized in the two-dimensional space.

We applied the linear function approximation versions of Differential Q-learning and RVI Q-learning on this problem. RVI Q-learning used the two reference functions discussed earlier in this section: (1) the action-value estimate corresponding to the *first* feature vector the agent observed when moving left, and (2) the maximum action value corresponding to the feature vectors *observed so far*, tracked online without storing all the previously observed feature vectors. Both algorithms used tile coding (Sutton & Barto 2018: Section 9.5.4) with 16 symmetric tilings of $2 \times 2 \times 2 \times 2 \times 2 \times 2$ tiles each. The weight vectors of both algorithms and the reward-rate estimate of Differential Q-learning was initialized to zero. The step-size parameter α was varied for both algorithms in the range $\{0.00125, 0.0025, 0.005, 0.01, 0.02, 0.04\}$. The parameter η for Differential Q-learning was varied in $\{0.1, 0.5, 1.0\}$. Each instance of parameters was applied for 10 runs of 400,000 time steps each. Both algorithms used an ϵ-greedy policy with $\epsilon = 0.1$ and no annealing.

The top-right panel of Figure 3.7 shows a typical learning curve on an instance of this problem where the goal positions is changed after every 100 time steps. Using

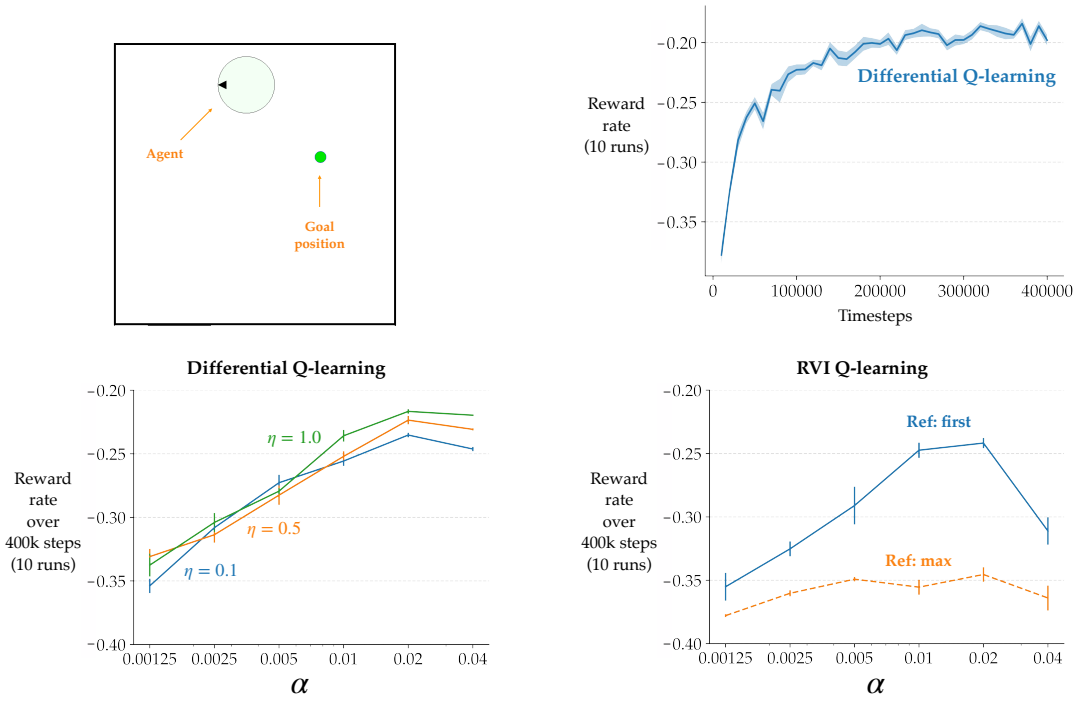


Figure 3.7: A learning curve and parameter studies for the linear function approximation versions of Differential Q-learning and RVI Q-learning on the PuckWorld problem. The shaded region and the error bars in the plots represent one standard error. *Top-left:* A still of the PuckWorld domain showing the agent and the goal position. *Top-right:* A typical learning curve started roughly at a reward rate of -0.4 and rose to about -0.19. *Bottom-left*: Parameter studies showing the performance of Differential Q-learning in terms of average reward rate was not very sensitive to the choice of η. *Bottom-right*: Parameter studies showing the performance of RVI Q-learning is relatively good when the reference function is the first observed feature vector, and relatively worse for the other reference function for a broad range of step sizes.

an ϵ-greedy policy with $\epsilon = 0.1$, the agent learns a policy that obtains a reward rate (computed over the last 10k steps) of about -0.19. The reward rate of a random policy is around -0.4. This learning curve corresponds to Differential Q-learning with $\alpha = 0.02, \eta = 1.0$. The learned policy was visualized and seen to be good everywhere except at the very edges of the two-dimensional space, which was probably an artifact of tile-coding.

We evaluated the performance of the agents across all the different parameter settings in terms of the average reward rate across the entire 400k time steps of

interaction. This is an indicator of the rate of learning. We observed that Differential Q-learning's rate of learning was quite robust to the parameter η. Its two parameters did not interact strongly; the best value of α was independent of the choice of η. Moreover, the best performance for different η values was roughly the same. These observations were similar to those in the tabular case.

RVI Q-learning also performed well on this problem for one choice of the reference function—the value estimate corresponding to the first feature vector the agent observes (with the 'left' action). The performance corresponding to the other reference function tested—tracking the maximum value of the observed feature vectors online—was not as good. This might be because unlike the tabular setting, updating the weights corresponding to one feature vector also modifies the estimate for other feature vectors, making the max hard to track. The best rate of learning corresponding to the better-performing reference function was slightly lower than that with Differential Q-learning.

We now move on to the second experiment in the linear function approximation setting. In the Catcher problem, the agent needs to catch as many falling fruits as possible. A still of the environment is shown in the top-left panel of Figure 3.8. 'Fruits' fall vertically down from a uniformly-random horizontal position starting at the top of the frame. The agent can control the position of a 'crate' at the bottom of the frame using two actions — left and right — which move the crate in that direction by a small amount. If the fruit falls on/in the crate, the agent gets a reward of +40; if the fruit falls anywhere outside at the bottom of the frame, the agent gets -40. The next fruit starts falling only after the previous fruit has reached the bottom of the frame. A fruit takes roughly 40 time steps to reach the the bottom starting from the top. Hence, the maximal reward rate on this problem is 1. At every time step, the agent observes a four-dimensional feature vector of the crate's horizontal position, the crate's horizontal velocity, the fruit's horizontal position, and the fruit's vertical position. The positions and velocity are scaled to lie roughly in $[0, 1]$ and in $[-1, 1]$

respectively.

All the experimental details are the same as for PuckWorld, the only difference being that both algorithms used tile coding with 8 symmetric tilings of $4 \times 4 \times 4 \times 4$ tiles each.

The top-right panel of Figure 3.8 shows a typical learning curve on this problem. Using an ϵ-greedy policy with $\epsilon = 0.1$, the agent learns a policy that obtains a reward

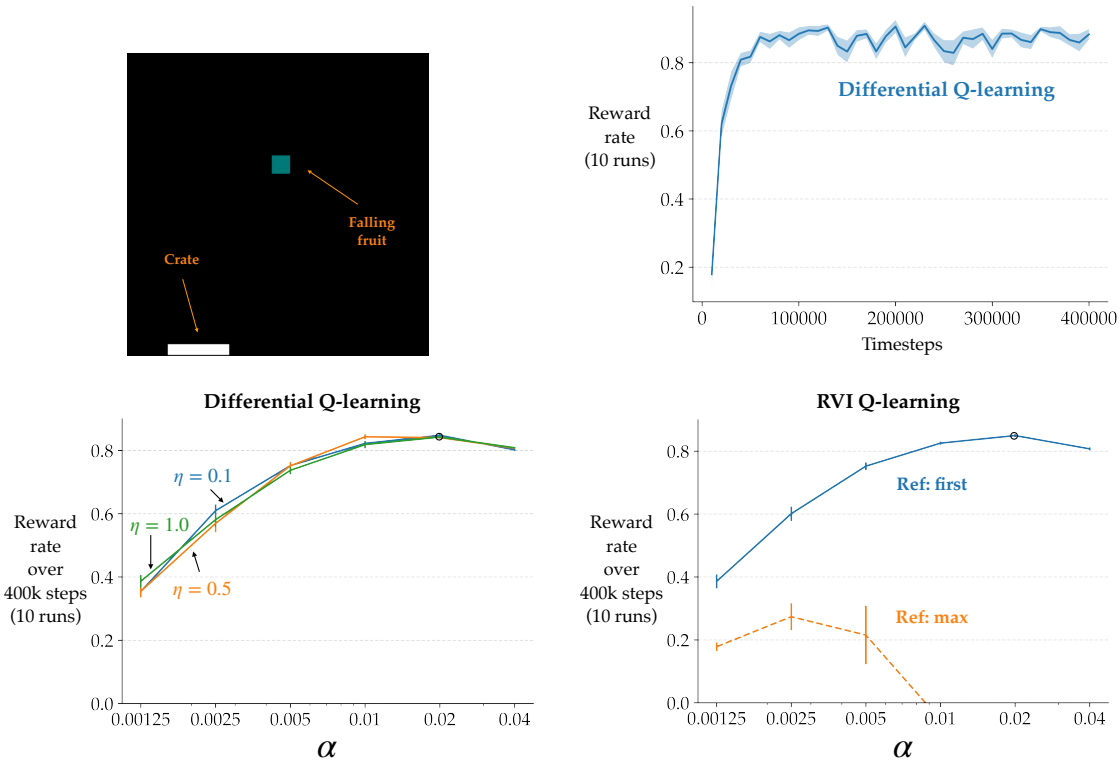


Figure 3.8: A learning curve and parameter studies for the linear function approximation versions of Differential Q-learning and RVI Q-learning on the Catcher problem. The shaded region and the error bars in the plots represent one standard error. *Top-left:* A still of the Catcher domain showing a falling fruit and the crate that the agent controls along the horizontal dimension at the bottom. *Top-right:* A typical learning curve started close to a reward rate of 0 and rose to about 0.9. *Bottom-left*: Parameter studies showing the performance of Differential Q-learning in terms of average reward rate was not very sensitive to the choice of parameters. *Bottom-right*: Parameter studies showing the performance of RVI Q-learning is relatively good when the reference function is the first observed feature vector, and relatively worse for the other reference function for a broad range of step sizes.

rate of about 0.85, which is close to the optimal reward rate of 1. The reward rate of a random policy is around -0.3. The learning curve shown corresponds to Differential Q-learning with $\alpha = 0.02, \eta = 1.0$.

Again, we evaluated the rate of the learning of the agents across different parameter settings. We again observed that Differential Q-learning's rate of learning did not vary much across a broad range of its parameter values. It was also especially robust to η. The linear function approximation version of RVI Q-learning also performed well for one choice of the reference function, not as much with the other. The learned policies corresponding to good parameter values for both algorithms successfully catch almost every fruit.

For RVI Q-learning, using the estimate of the first observed feature vector as a reference value worked better in Catcher than in PuckWorld. This might be because the agent might be observing feature vectors similar to the first one quite frequently, given that the crate has to move across the whole one-dimensional horizontal plane under any optimal policy. On the other hand, the agent moves in a relatively larger two-dimensional space in PuckWorld. In a finite number of agent-environmental interactions, the agent might not visit its starting location that frequently. This suggests that the choice of the reference feature vector can affect the performance of RVI Q-learning differently in different problems. Additionally, in both cases, the other reference function did not result in good performance; this was probably because tracking the maximum action value in the function approximation setting is a poor approximation to the maximum action value across all state–action pairs.

The two experiments showed that the simple extension of the tabular Differential Q-learning to the linear function approximation setting can work rather well in terms of the final performance as well as robustness to different parameter values. The extension of the notion of reference functions to the linear function approximation setting is not as straightforward.

3.5 Discussion and Conclusion

In this chapter, I proposed one-step average-reward learning algorithms for on- and off-policy prediction and control.

- The key idea behind the Differential algorithms is the explicit estimation of the average reward of the target policy, in particular, with an update that uses the TD error instead of, say, the sample average of the observed rewards. As a result, Differential TD-learning and Differential Q-learning are applicable in the off-policy setting and do not require any special reference function.

- Through a series of experiments, I found that Differential TD-learning and Differential Q-learning lead to good performance for a large range of their parameter values, making them relatively easy to use.

Together, the two algorithms improve the generality and applicability of average-reward methods, and hence hold great promise for average-reward reinforcement learning.

An important way in which the work in this chapter is limited is that it primarily treats the tabular case, with some extensions to linear function approximation. The tabular case is important to build intuitions and understand the ideas deeply, but some form of function approximation is necessary for large-scale applications and the larger ambitions of artificial intelligence. Indeed, the need for function approximation is a large part of the motivation for studying the average-reward setting. In this dissertation, we take a few steps in this direction. In Chapter 4, we extend the theory of Differential TD-learning to the case of linear function approximation. Further, in Chapter 5, we (indirectly) assess variants of Differential Q-learning with both linear and non-linear function approximation.

Besides handling spatial abstraction via function approximation, it is also important to extend these algorithms to handle temporal abstraction. Yi Wan, Rich Sutton,

and I extended the options framework (Sutton et al., 1999a) for temporal abstraction from discounted MDPs to average-reward MDPs. The work was led by Yi and is part of his Ph.D. dissertation (Wan, 2023).

Finally, the algorithms in this chapter are also limited to one-step updates. Multi-step algorithms have been shown to propagate information faster over longer temporal spans that one-step algorithms (see, e.g., Sutton & Barto's (2018) Chapters 7 and 12). In the next chapter we present the multi-step versions of Differential TD-learning that use eligibility traces for efficient and online updates.

Chapter 4

Multi-step Differential Methods for Prediction with Eligibility Traces

A discussion of foundational average-reward algorithms for RL is incomplete without multi-step algorithms. Multi-step algorithms have been shown to propagate information faster over longer temporal spans than one-step algorithms (see, e.g., Sutton & Barto, 2018). Eligibility traces are typically used to implement multi-step updates online and independent of the temporal span of predictions (Sutton, 1988a; van Hasselt & Sutton, 2015).

This chapter presents the first multi-step average-reward prediction algorithm that is guaranteed to converge in the off-policy case. Tsitsiklis and Van Roy (1999) took the first and biggest step in developing the first multi-step algorithm for the average-reward formulation. They established the convergence of their Average-Cost TD(λ) algorithm for on-policy *linear function approximation* at a time when there was no multi-step prediction result even in the tabular case. They achieved this by cleverly extending proof techniques that they had recently developed for the discounted-reward formulation (Tsitsiklis and Van Roy, 1997).

However, Average-Cost TD(λ) is limited to the on-policy case. It cannot be extended to the off-policy case because the update for the average-reward estimate does not generalize to the off-policy setting (this was also explained in Chapter 3's Section 3.1). Differential TD uses a different average-reward update that is proven to

converge appropriately in the off-policy case, which motivates an extension to the multi-step case.

In this chapter, I present multi-step versions of the tabular Differential TD-learning methods discussed in Chapter 3. In particular, I present two prediction algorithms that use eligibility traces, analyze their stability and convergence, and present empirical results that validate the theoretical results. The two algorithms are two attempts to develop a single algorithm that can be proved to converge for both the on- and off-policy settings in the tabular case of the average-reward formulation:

1. Algorithm 1 converges in the on-policy setting even with linear function approximation.

2. Algorithm 2 belongs to a family of algorithms that converges in the off-policy setting in the tabular case.

Algorithm 1 and 2 have subtle differences. Algorithm 1 is restricted to the on-policy setting; I show that a simple extension of it to the off-policy setting can diverge even in the tabular case. Before diving into the algorithms, I introduce some mathematical notation.

4.1 Multi-step Notation

Throughout the previous chapter, we considered one-step algorithms. That is, the information in the single transition at time step t—$(S_t, A_t, S_{t+1}, R_{t+1})$—is used to update the value estimates $\mathbf{v}_t(S_t)$ by creating a target: $R_{t+1} - r(\pi) + \mathbf{v}_t(S_{t+1})$, where $\mathbf{v}_t \in \mathbb{R}^{|\mathcal{S}|}$ denotes the vector of value estimates at time step t. Such a target constructed from the information of one time step is called a one-step target, which we denote by $G_{t:t+1}$.

Just like how Sutton and Barto (2018) defined general n-step targets for the discounted-reward formulation, we can define n-step targets $G_{t:t+n}$ for the average-

reward formulation using data from n transitions:

$$G_{t:t+n} \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + \ldots + R_{t+n} - r(\pi) + \mathbf{v}_t(S_{t+n}).$$

The λ-return for the average-reward formulation is also defined with an exponentially decaying weight per step, parameterized by $\lambda \in [0, 1)$:

$$\begin{aligned} G_t^\lambda &\doteq (1-\lambda)\big[G_{t:t+1} + \lambda G_{t:t+2} + \lambda^2 G_{t:t+3} + \ldots\big] \\ &= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}, \end{aligned}$$

with the $(1-\lambda)$ factor ensuring that the weights sum to one.

Iteratively updating the estimates based on such targets obtained from samples of experience is the basis of sample-based algorithms. In the tabular case, these *model-free* RL algorithms exactly estimate the true value function and the reward rate if the noise in the individual targets is averaged out slowly.

If a full model of the reward and transition dynamics is available, we can construct targets based on the expected outcomes of all possible transitions. As a result, for instance, the value estimates of all the states can be updated together. For one-step targets, the expected updates for a given reward rate $\bar{r}$ and policy π is:

$$\begin{aligned} \mathbf{v}_{t+1}(s) &\doteq \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r - \bar{r} + \mathbf{v}_t(s')\big], \quad \forall s, \\ \text{or,} \quad \mathbf{v}_{t+1} &\doteq \mathbf{r}_\pi - \bar{r}\mathbf{1} + \mathbf{P}_\pi \mathbf{v}_t = T_\pi^1 \mathbf{v}_t, \end{aligned}$$

where $\mathbf{r}_\pi$ denotes the expected one-step reward when taking actions from π, $\mathbf{P}_\pi$ denotes the state-to-state transition matrix of the Markov chain induced by π, and $T_\pi^1 : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ is a linear transformation matrix. The iterative update resembles the average-reward Bellman equation (2.6) and T_π^1 is aptly called the one-step Bellman operator.

The operators corresponding to n-step targets and the λ-return respectively are:

$$
\begin{aligned}
T_\pi^n \mathbf{v} &\doteq \sum_{k=1}^{n} \mathbf{P}_\pi^{k-1}(r_\pi - \bar{r}\mathbf{1}) + \mathbf{P}_\pi^n \mathbf{v}, \\
T_\pi^\lambda \mathbf{v} &\doteq (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} T_\pi^n \mathbf{v}, \\
&= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\Big(\sum_{k=1}^{n} \mathbf{P}_\pi^{k-1}(r_\pi - \bar{r}\mathbf{1}) + \mathbf{P}_\pi^n \mathbf{v}\Big).
\end{aligned}
$$

Again, as in the discounted-reward formulation, iteratively applying the Bellman operators leads a sequence of estimates $\mathbf{v}_1, \mathbf{v}_2, \ldots$ that eventually converge to $\mathbf{v}_\infty = \mathbf{v}$, which is a solution of the Bellman equations. Consider the example with the operator for the λ-return, which we call the *multi-step Bellman operator*. After a sequence of updates:

$$
\begin{aligned}
T_\pi^\lambda \mathbf{v} &= \mathbf{v} && (4.1)\\
\text{or,}\quad \mathbf{v} &= T_\pi^\lambda \mathbf{v} \\
&= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\Big(\sum_{k=1}^{n} \mathbf{P}_\pi^{k-1}(r_\pi - \bar{r}\mathbf{1}) + \mathbf{P}_\pi^n \mathbf{v}\Big) \\
&= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\Big(\sum_{k=1}^{n} \big(\mathbf{P}_\pi^{k-1}\mathbf{r}_\pi - \bar{r}\,\mathbf{1}\big) + \mathbf{P}_\pi^n \mathbf{v}\Big) \\
&= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\Big(\sum_{k=1}^{n} \big(\mathbf{P}_\pi^{k-1}\mathbf{r}_\pi - \bar{r}\,\mathbf{1}\big)\Big) + (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\mathbf{P}_\pi^n \mathbf{v} \\
&= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\Big(\sum_{k=1}^{n} \big(\mathbf{P}_\pi^{k-1}\mathbf{r}_\pi - \bar{r}\,\mathbf{1}\big)\Big) + \mathbf{P}_\pi^\lambda \mathbf{v} && (4.2)\\
&= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\Big(\sum_{k=1}^{n} \big(\mathbf{P}_\pi^{k-1}\mathbf{r}_\pi\big) - n\,\bar{r}\,\mathbf{1}\Big) + \mathbf{P}_\pi^\lambda \mathbf{v} \\
&= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1}\Big(\sum_{k=1}^{n} \mathbf{P}_\pi^{k-1}\mathbf{r}_\pi\Big) - \frac{\bar{r}}{1-\lambda}\,\mathbf{1} + \mathbf{P}_\pi^\lambda \mathbf{v} \quad \text{(sum of AGP)} \\
&= \sum_{n=1}^{\infty} \lambda^{n-1}\mathbf{P}_\pi^{n-1}\mathbf{r}_\pi - \frac{\bar{r}}{1-\lambda}\,\mathbf{1} + \mathbf{P}_\pi^\lambda \mathbf{v} \\
\mathbf{v} &= \sum_{k=0}^{\infty} \lambda^{k}\mathbf{P}_\pi^{k}\mathbf{r}_\pi - \frac{\bar{r}}{1-\lambda}\,\mathbf{1} + \mathbf{P}_\pi^\lambda \mathbf{v}, && (4.3)
\end{aligned}
$$

where AGP refers to an arithmetic-geometric progression, and the second-to-last

equality is a result of a nifty rearrangement of the terms of summation. (4.3) is a form of the multi-step Bellman equation for the average-reward formulation. Note that $\lambda = 0$ gives the familiar one-step Bellman equation (2.6). Further, recall that the average-reward Bellman equation has infinite solutions for $(\mathbf{v}, \bar{r})$ of the form $(\mathbf{v}_\pi + c\mathbf{1}, r(\pi))$ in unichain MDPs, where $c \in \mathbb{R}$ and $\mathbf{v}_\pi$ denotes the differential state-value function corresponding to a policy π and $r(\pi)$ denotes the average reward per step obtained using π.

4.2 On-policy: Theory

We begin with the presentation of the on-policy multi-step prediction algorithm. It extends Chapter 3's one-step tabular Differential TD-learning in two ways:

1. it estimates the *multi-step* λ-return online and incrementally,
2. it is proved to converge with *linear function approximation.*

We first introduce the terminology for function approximation and the special case of linear function approximation. In Chapter 3 we dealt with the tabular case in which the true value function can be approximated exactly with a parameter per state or state–action pair. In general, this is not possible. The world in which every learning agent resides is much larger than any individual agent. So in general the agent does not observe the true state of the world, but just a tiny fraction of it through its sensory inputs. Mathematically, if the true state of the world evolves in state space $\mathcal{S}$, an agent just observes a vector $\mathbf{x} \in \mathbb{R}^d$ where $d << |\mathcal{S}|$. Based on this small observation vector, the agent approximates the value function with a set of parameters $\mathbf{w} \in \mathbb{R}^l$. In the general function approximation setting, the approximate values can never exactly match the true values. The goal of the prediction problem is then to estimate the true values as closely as possible.

Linear function approximation is a kind of function approximation where the estimated values are linear function of the parameters (also called the weight vector) $\mathbf{w}$.

In particular, corresponding to an observation vector $\mathbf{x}$, we consider a linear value estimate of the form $\mathbf{w}^\top \mathbf{x}$, where $l = d << |\mathcal{S}|$. For simplicity, we assume that *we* know the exact state of the world at all times but the agent only has access to a small portion of it. That is, for every state $s \in \mathcal{S}$, the agent observes $\mathbf{x}(s) \in \mathbb{R}^d$, and the corresponding value estimate is $\hat{v}(s) = \mathbf{w}^\top \mathbf{x}(s)$.

Convergence Theorem

We are now ready to understand the on-policy linear case. I first present the algorithm. Its analysis is based on the Tsitsiklis and Van Roy's (1999) analysis of their Average-Cost TD(λ) algorithm, so we also follow the same assumptions.

Assumption 4.1. *The Markov chain corresponding to the target policy is irreducible and aperiodic.*

This ergodicity assumption simplifies the presentation. Extension to the more general unichain case would require mild modifications to some of the analysis and a specification of convergence to states that are recurrent under the target policy. For the most general multichain case, we would have to repeat the same analysis for every chain. Aperiodicity is important for the unique existence of a steady-state distribution over states.

It follows that there is a unique stationary distribution $\mathbf{d}_\pi$ and a unique reward rate $r(\pi)$ for the Markov chain induced by the target policy π:

$$r(\pi) = \sum_s d_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\ r.$$

Let $\mathbf{X}$ denotes a $|\mathcal{S}| \times d$ matrix whose i^{th} row is the d-dimensional feature vector corresponding to state $s_i \in \mathcal{S}$. Π denotes a projection matrix that projects a vector onto the subspace spanned by the feature vectors: $\Pi = \mathbf{X}(\mathbf{X}^\top \mathbf{D}_\pi \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{D}_\pi$, where $\mathbf{D}_\pi$ is a $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix with elements of $\mathbf{d}_\pi$ along the diagonal. And let $\|\cdot\|$ denote the Euclidean norm for vectors — $\|\mathbf{v}\| \doteq \sqrt{\mathbf{v}^\top \mathbf{v}}$ — and the Euclidean-induced norm on matrices — $\|\mathbf{M}\| \doteq \max_{\mathbf{v} \ni \|\mathbf{v}\|=1} \|\mathbf{M}\mathbf{v}\|$.

Assumption 4.2. *Features in* $\mathbf{X}$ *are linearly independent and bounded. Additionally,* $\mathbf{X}\mathbf{w} \neq \mathbf{1}$ *for any* $\mathbf{w} \in \mathbb{R}^d$.

The features need to be linearly independent for the linear-algebra-based analysis to work; their boundedness implies the boundedness of the trace vector, which is important for the analysis. The second part of the assumption basically means that the constant vector is not representable by the linear value approximator. Tsitsiklis and Van Roy used this to eliminate the possibility of multiple solutions (recall that the differential Bellman equations have infinite solutions).

Our first algorithm is a straightforward extension of the one-step tabular on-policy Differential TD-learning. Recall from Chapter 3 that on-policy Differential TD-learning updates its tabular estimates $V : \mathcal{S} \to \mathbb{R}$ and its reward-rate estimate $\bar{R} \in \mathbb{R}$ after every transition $(S_t, A_t, S_{t+1}, R_{t+1})$ as:

$$\begin{aligned} V_{t+1}(S_t) &\doteq V_t(S_t) + \alpha_t \delta_t, \\ \bar{R}_{t+1} &\doteq \bar{R}_t + \eta \alpha_t \delta_t, \\ \text{where,} \quad \delta_t &= R_{t+1} - \bar{R}_t + V_t(S_{t+1}) - V_t(S_t). \end{aligned}$$

On-policy multi-step linear Differential TD-learning has an additional trace vector $\mathbf{z} \in \mathbb{R}^d$ which is updated at every step along with the weight vector $\mathbf{w} \in \mathbb{R}^d$ and the reward-rate estimate $\bar{R}$ (see Appendix A for the complete pseudocode).

Linear on-policy Differential TD(λ) (Algorithm 1)

After observing $\mathbf{x}_t$, taking A_t according to π, observing $\mathbf{x}_{t+1}$ and R_{t+1}:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha_t \delta_t \mathbf{z}_t, \tag{4.4}$$

$$\bar{R}_{t+1} \doteq \bar{R}_t + \eta \alpha_t \delta_t, \tag{4.5}$$

$$\text{where,} \quad \mathbf{z}_t \doteq \lambda \mathbf{z}_{t-1} + \mathbf{x}_t, \tag{4.6}$$

$$\text{and,} \quad \delta_t \doteq R_{t+1} - \bar{R}_t + \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t. \tag{4.7}$$

For the convergence theorem, we need the standard assumption for stochastic-approximation algorithms regarding step sizes.

Assumption 4.3. *The step sizes α_t are positive, deterministic, and satisfy $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$.*

We are now ready for the convergence theorem.

Theorem 4.1. *Under Assumptions 4.1, 4.2, 4.3, on-policy linear Differential TD(λ) (Algorithm 1) converges for all $\lambda \in [0, 1)$ with probability one:*

1. *$\bar{R}$ converges to the unique reward rate of the target policy $r(\pi)$.*
2. *$\mathbf{w}$ converges to the unique solution, $\mathbf{w}^*$, of $\Pi T^\lambda(\mathbf{X}\mathbf{w}) = \mathbf{X}\mathbf{w}$.*

The following error bound holds w.r.t. the centered differential value function $\mathbf{v}_\pi$:

$$\inf_{c \in \mathbb{R}} \|\mathbf{X}\mathbf{w}^* - (\mathbf{v}_\pi + c\,\mathbf{1})\|_{\mathbf{d}_\pi} \leq \frac{1}{\sqrt{1 - \tau_\lambda^2}} \inf_{c \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d} \|\mathbf{X}\mathbf{w} - (\mathbf{v}_\pi + c\,\mathbf{1})\|_{\mathbf{d}_\pi},$$

where τ_λ is a function of λ such that $\tau_\lambda \in [0, 1)$ and $\lim_{\lambda \to 1} \tau_\lambda = 0$;

Here, $\|\cdot\|_{\mathbf{p}}$ denotes the weighted Euclidean norm for vectors — $\|\mathbf{v}\|_{\mathbf{p}} \doteq \sqrt{\sum_i \mathbf{p}_i \mathbf{v}_i^2}$.

Convergence Proof

The convergence proof is based on Tsitsiklis and Van Roy's (1999) analysis of their Average-Cost TD(λ) algorithm, which is also an on-policy multi-step average-reward prediction algorithm. We discussed how its one-step version differs from our differential style of algorithms in Chapter 3. To reiterate, the two algorithms differ in how the reward-rate estimate is updated: Differential TD(λ) updates it using the TD error while Average-Cost TD(λ) maintains an exponential-recency-weighted average of the observed rewards.

Linear on-policy Average-Cost TD(λ)(Tsitsiklis and Van Roy, 1999)

After observing $\mathbf{x}_t$, taking A_t according to π, observing $\mathbf{x}_{t+1}$ and R_{t+1}:

$$\begin{aligned}
\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha_t(R_{t+1} - \bar{R}_t + \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t)\mathbf{z}_t, \\
\bar{R}_{t+1} &\doteq \bar{R}_t + \eta\alpha_t(R_{t+1} - \bar{R}_t), \\
\mathbf{z}_{t+1} &\doteq \lambda \mathbf{z}_t + \mathbf{x}_t.
\end{aligned}$$

Our analysis mirrors that of Tsitsiklis and Van Roy's for Average-Cost TD(λ). In particular, we use the ODE approach to show that the sequence of estimates generated by the algorithm resembles the solution to an ODE which has a unique asymptotically stable equilibrium point. For this purpose, Tsitsiklis and Van Roy used a specific instance of a general stochastic-approximation result and showed that Average-Cost TD(λ) satisfies all the conditions for the result to apply. We do the same for Algorithm 1.

We begin by rewriting the update equations as:

$$\bar{R}_{t+1} \doteq \bar{R}_t + \eta\alpha_t\big[R_{t+1} - \bar{R}_t \ - (\mathbf{x}_t - \mathbf{x}_{t+1})^\top \mathbf{w}_t\big], \tag{4.8}$$

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t \ + \mathbf{z}_t\alpha_t\big[R_{t+1} - \bar{R}_t - (\mathbf{x}_t - \mathbf{x}_{t+1})^\top \mathbf{w}_t\big] \tag{4.9}$$

Now we can combine the learnable parameters $\bar{R} \in \mathbb{R}$ and $\mathbf{w} \in \mathbb{R}^d$ into a single vector $\mathbf{u} \in \mathbb{R}^{d+1}$ for a more compact notation, such that $\mathbf{u}_0 = \bar{R}$ and $\mathbf{u}_{1:d} = \mathbf{w}$:

$$\mathbf{u}_t \doteq \begin{bmatrix} \bar{R}_t \\ \mathbf{w}_t(1) \\ \vdots \\ \mathbf{w}_t(d) \end{bmatrix}_{(d+1)\times 1} = \begin{bmatrix} \bar{R}_t \\ \mathbf{w}_t \end{bmatrix}_{(d+1)\times 1}.$$

The updates can now be written as:

$$\mathbf{u}_{t+1} \doteq \mathbf{u}_t + \alpha_t\big(\mathbf{b}(Y_t) + \mathbf{A}(Y_t)\mathbf{u}_t\big), \tag{4.10}$$

where, $Y_t \doteq (\mathbf{z}_t, S_t, A_t, R_{t+1}, S_{t+1})$,

$$\mathbf{b}(Y_t) = \mathbf{b}_t \doteq \begin{bmatrix} \eta R_{t+1} \\ \mathbf{z}_t R_{t+1} \end{bmatrix}_{(d+1)\times 1}, \tag{4.11}$$

$$\mathbf{A}(Y_t) = \mathbf{A}_t \doteq \begin{bmatrix} -\eta & \eta(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \\ -\mathbf{z}_t & \mathbf{z}_t(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \end{bmatrix}_{(d+1)\times(d+1)}. \tag{4.12}$$

$\{Y_t\}$ is a Markov chain: the next state and reward are determined by the current state and action, and the trace vector evolves as $\mathbf{z}_{t+1} = \lambda \mathbf{z}_t + \mathbf{x}(S_{t+1})$. Note that in this on-policy setting each element of the trace vector is bounded (by $x_{\max}/(1-\lambda)+k$, where $x_{\max}$ is the maximum absolute feature value and k is the largest element of the trace vector's initialization: $\mathbf{z}_{-1} \in \mathbb{R}^d$). Since $\mathcal{S}, \mathcal{A}, \mathcal{R}$ are finite discrete sets, the chain $\{Y_t\}$ evolves in a closed and bounded space $\mathcal{Y}$. Based on Yu's (2012, 2017) analysis, $\{Y_t\}$ has a unique invariant probability measure d.

A sufficient (but not necessary) condition for stability is that matrix $\mathbf{A}$ should be negative definite[1], where $\mathbf{A}$ denotes the long-term expected value of $\mathbf{A}_t$ according to d: $\mathbf{A} = \mathbb{E}_d[\mathbf{A}_t]$. We now derive the expression for $\mathbf{A}$ and show that it is indeed a negative definite matrix.

Like Tsitsiklis and Van Roy (1999), we construct an alternative stationary Markov chain to compute the expected values instead of taking the limit of the expectations as t goes to infinity. Consider a double-ended stationary Markov chain $\{Y_t\}$ evolving in $\mathcal{Y}$ such that for integers $t \in (-\infty, \infty)$, $\Pr(\mathbf{z}_t \in \mathcal{B}, S_t = s, A_t = a, S_{t+1} = s') = d(\mathcal{B}, s, a, s')$, where $\mathcal{B}$ is any Borel subset of $\mathcal{Y}$. Let $\mathbb{E}_d$ also denote the expectations w.r.t. the double-ended stationary Markov chain.

Lemma 4.1. *Under Assumption 4.1, the following hold:*

1. $\mathbb{E}_d[R_{t+1}] = r(\pi)$
2. $\mathbb{E}_d[\mathbf{x}_t] = \mathbf{X}^\top \mathbf{D}_\pi \mathbf{1}$
3. $\mathbb{E}_d[\mathbf{z}_t] = \frac{1}{1-\lambda}\mathbf{X}^\top \mathbf{D}_\pi \mathbf{1}$

[1]Strictly speaking, only symmetric matrices can be positive/negative-definite. Here, we use the term loosely to imply the same property for non-symmetric matrices.

4. $\mathbb{E}_d[\mathbf{x}_{t-k} R_{t+1}] = \mathbf{X}^\top \mathbf{D}_\pi^k \mathbf{r}_\pi$

5. $\mathbb{E}_d[\mathbf{z}_t R_{t+1}] = \sum_{k=0}^{\infty} \lambda^k \mathbf{X}^\top \mathbf{D}_\pi^k \mathbf{r}_\pi$

6. $\mathbb{E}_d[\mathbf{x}_{t-k} \mathbf{x}_t^\top] = \mathbf{X}^\top \mathbf{D}_\pi \mathbf{P}_\pi^k \mathbf{X}$

7. $\mathbb{E}_d[\mathbf{z}_t \, \mathbf{x}_t^\top] = \sum_{k=0}^{\infty} \lambda^k \mathbf{X}^\top \mathbf{D}_\pi \mathbf{P}_\pi^k \mathbf{X}$

8. $\mathbb{E}_d[\mathbf{z}_t (\mathbf{x}_{t+1} - \mathbf{x}_t)^\top] = \sum_{k=0}^{\infty} \lambda^k \mathbf{X}^\top \mathbf{D}_\pi \mathbf{P}_\pi^k (\mathbf{P}_\pi - \mathbb{I}) \mathbf{X}$

9. $\mathbb{E}_d[\mathbf{x}_{t+1} - \mathbf{x}_t] = 0$

Proof. 1. $\mathbb{E}_d[R_{t+1}] = \sum_s d_\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \; r = r(\pi)$, by definition.

2. $\mathbb{E}_d[\mathbf{x}_t] = \mathbb{E}_d[\mathbf{x}(S_t)] = \sum_s d_\pi(s) \mathbf{x}(s) = \mathbf{X}^\top \mathbf{d}_\pi = \mathbf{X}^\top \mathbf{D}_\pi \mathbf{1}$

3. Note that the expressions are the same for any time step t because they are expectations of functions of Y under the invariant probability measure d. So without loss of generality, we compute the expressions for $t = 0$. Now, based on Yu's (2012, 2017) analysis, we can write $\mathbf{z}_0 = \sum_{k=-\infty}^{0} \lambda^{-k} \mathbf{x}_k$ almost surely, where $\mathbf{x}_k$ denotes the feature vector at the k-th time step. Next, note that the trace parameter does not affect the evolution of the original Markov chain $\{S_t\}$ induced by the behavior policy b, hence the marginal distribution of states in Y is the same as $\mathbf{d}_b$ from the original Markov chain.

$$
\begin{aligned}
\mathbb{E}_d[\mathbf{z}_0] &= \mathbb{E}_d\Big[\sum_{k=-\infty}^{0} \lambda^{-k} \mathbf{x}_k \Big] \\
&= \sum_{k=-\infty}^{0} \lambda^{-k} \mathbb{E}_d[\mathbf{x}_k] \\
&= \sum_{k=0}^{\infty} \lambda^k \mathbf{X}^\top \mathbf{D}_\pi \mathbf{1} \\
&= \frac{1}{1-\lambda} \mathbf{X}^\top \mathbf{D}_\pi \mathbf{1}.
\end{aligned}
$$

4.

$$\begin{aligned}\mathbb{E}_d[\mathbf{x}_{t-k}R_{t+1}] &= \sum_s d_\pi(s)\sum_{s'}\Pr(s'|s,k,\pi)\sum_{a'}\pi(a'|s')\sum_{r'}p(r'|s',a')[r'\,\mathbf{x}(s)] \\ &= \sum_s d_\pi(s)\mathbf{x}(s)\sum_{s'}\Pr(s'|s,k,\pi)\,r_\pi(s') \\ &= \mathbf{X}^\top\mathbf{D}_\pi\mathbf{P}_\pi^k\mathbf{r}_\pi\end{aligned}$$

5. As in part 3, we can consider the case when $t = 0$:

$$\begin{aligned}\mathbb{E}_d[\mathbf{z}_0R_1] = \mathbb{E}_d[\sum_{k=-\infty}^{0}\lambda^{-k}\mathbf{x}_kR_1] &= \mathbb{E}_d[\sum_{k=0}^{\infty}\lambda^k\mathbf{x}_{-k}R_1] \\ &= \sum_{k=0}^{\infty}\lambda^k\mathbb{E}_d[\mathbf{x}_{-k}R_1] \\ &= \sum_{k=0}^{\infty}\lambda^k\mathbf{X}^\top\mathbf{D}_\pi\mathbf{P}_\pi^k\mathbf{r}_\pi\end{aligned}$$

6.

$$\begin{aligned}\mathbb{E}_d[\mathbf{x}_{t-k}\,\mathbf{x}_t^\top] &= \sum_s d_\pi(s)\sum_{s'}\Pr(s'\mid s,k,\pi)\,\mathbf{x}(s)\,\mathbf{x}(s')^\top \\ &= \mathbf{X}^\top\mathbf{D}_\pi\mathbf{P}_\pi^k\mathbf{X}\end{aligned}$$

7. Again, we consider the case when $t = 0$: $\mathbb{E}_d[\mathbf{z}_t\,\mathbf{x}_t^\top] = \mathbb{E}_d[\sum_{k=-\infty}^{0}\lambda^{-k}\mathbf{x}_k\mathbf{x}_0^\top] = \mathbb{E}_d[\sum_{k=0}^{\infty}\lambda^k\mathbf{x}_{-k}\mathbf{x}_0^\top] = \sum_{k=0}^{\infty}\lambda^k\mathbb{E}_d[\mathbf{x}_{-k}\mathbf{x}_0^\top] = \sum_{k=0}^{\infty}\lambda^k\mathbf{X}^\top\mathbf{D}_\pi\mathbf{P}_\pi^k\mathbf{X}$

8.

$$\begin{aligned}\mathbb{E}_d[\mathbf{z}_t(\mathbf{x}_{t+1}-\mathbf{x}_t)^\top] &= \mathbb{E}_d[\mathbf{z}_t\mathbf{x}_{t+1}^\top] - \mathbb{E}_d[\mathbf{z}_t\mathbf{x}_t^\top] \\ &= \sum_{k=0}^{\infty}\lambda^k\mathbf{X}^\top\mathbf{D}_\pi\mathbf{P}_\pi^{k+1}\mathbf{X} - \sum_{k=0}^{\infty}\lambda^k\mathbf{X}^\top\mathbf{D}_\pi\mathbf{P}_\pi^k\mathbf{X} \\ &= \sum_{k=0}^{\infty}\lambda^k\mathbf{X}^\top\mathbf{D}_\pi\mathbf{P}_\pi^k(\mathbf{P}_\pi-\mathbb{I})\mathbf{X}\end{aligned}$$

9. $\mathbb{E}_d[\mathbf{x}_t - \mathbf{x}_{t+1}] = \mathbb{E}_d[\mathbf{x}_t] - \mathbb{E}_d[\mathbf{x}_{t+1}] = 0$

□

Lemma 4.2. *Under Assumption 4.1, the steady-state expectations* $\mathbf{b} = \mathbb{E}_d[\mathbf{b}_t]$ *and* $\mathbf{A} = \mathbb{E}_d[\mathbf{A}_t]$ *are given by:*

$$\mathbf{b} = \begin{bmatrix} \eta \; r(\pi) \\ \sum_{k=0}^{\infty} \lambda^k \mathbf{X}^\top \mathbf{D}_\pi \mathbf{P}_\pi^k \mathbf{r}_\pi \end{bmatrix}_{(d+1)\times 1} \tag{4.13}$$

$$\mathbf{A} = \begin{bmatrix} -\eta & \mathbf{0}^\top \\ \frac{-1}{1-\lambda}\mathbf{X}^\top \mathbf{D}_\pi \mathbf{1} & \sum_{k=0}^{\infty} \lambda^k \mathbf{X}^\top \mathbf{D}_\pi \mathbf{P}_\pi^k (\mathbf{P}_\pi - \mathbb{I})\mathbf{X} \end{bmatrix}_{(d+1)\times(d+1)} \tag{4.14}$$

Proof. $\mathbf{b}$ and $\mathbf{A}$ are given by:

$$\mathbf{b} = \begin{bmatrix} \mathbb{E}_d[\eta R_{t+1}] \\ \mathbb{E}_d[\mathbf{z}_t R_{t+1}] \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} \mathbb{E}_d[-\eta] & \mathbb{E}_d[\eta(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top] \\ \mathbb{E}_d[-\mathbf{z}_t] & \mathbb{E}_d[\mathbf{z}_t(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top] \end{bmatrix}$$

The result follows from Lemma 4.1. □

It turns out that the expression for $\mathbf{A}$ corresponding to Algorithm 1 is the same as that of Average-Cost TD(λ), despite the expressions of $\mathbf{A}_t$ being different.

So we can directly apply their Lemma 7 to show the negative definiteness of $\mathbf{A}$.

Lemma 4.3. *There exists a diagonal matrix* $\mathbf{L}$ *with positive diagonal entries such that* $\mathbf{LA}$ *is negative definite.*

Proof. Because $\mathbf{A}$ here is the same as Tsitsiklis and Van Roy's (1999), the result follows from their Lemma 7. □

Tsitsiklis and Van Roy scale the matrix $\mathbf{A}$ with a diagonal matrix $\mathbf{L}$ having the first element as some positive number $l > 0$ and the other diagonal entries as one. This is because for every $\mathbf{A}$ of the above form, there exists a sufficiently large l such that $\mathbf{LA}$ is negative definite.

Lemma 4.4. *There exists a constant* C *such that for all* $t \geq 0$ *in* $\{Y_t\}$ *starting with any* $S_0 \in \mathcal{S}$ *and* $\mathbf{z}_{-1} \in \mathbb{R}^d$*:*

1. $\|\mathbf{b}(Y_t)\| \leq C$
2. $\|\mathbf{A}(Y_t)\| \leq C$

Proof. Given S_0 and $\mathbf{z}_{-1}$, S_t evolves according to $\mathbf{P}_\pi$, resulting in S_{t+1}, R_{t+1}, and $\mathbf{z}_t = \lambda \mathbf{z}_{t-1} + \mathbf{x}(S_t)$.

1. We know that:

$$\mathbf{b}(Y_t) \doteq \begin{bmatrix} \eta R_{t+1} \\ \mathbf{z}_t R_{t+1} \end{bmatrix}.$$

   Then,

$$\begin{aligned} \|\mathbf{b}(Y_t)\| &= \sqrt{\eta^2 R_{t+1}^2 + \mathbf{z}_t(1)^2 R_{t+1}^2 + \ldots + \mathbf{z}_t(d)^2 R_{t+1}^2} \\ &= |R_{t+1}| \sqrt{\eta^2 + \mathbf{z}_t(1)^2 + \ldots + \mathbf{z}_t(d)^2} \\ &\leq R_{\max} \sqrt{\eta^2 + d(\frac{1}{1-\lambda} x_{\max}^2 + k^2)} = C_1 \end{aligned}$$

   where $R_{\max}$ is the maximum absolute value of rewards, $x_{\max}$ is the maximum absolute feature value, and k is the largest element of $\mathbf{z}_{-1}$.

2. Next,

$$\mathbf{A}(Y_t) \doteq \begin{bmatrix} -\eta & \eta(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \\ -\mathbf{z}_t & \mathbf{z}_t(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \end{bmatrix}.$$

   $\|\mathbf{A}(Y_t)\| = \max_{\mathbf{v} \ni \|\mathbf{v}\|=1} \|\mathbf{A}(Y_t)\mathbf{v}\|$. Let $\mathbf{v}^*$ be the vector in the subspace $\|\mathbf{v}\| = 1$ for which the maximum is obtained. Then $\mathbf{A}(Y_t)\mathbf{v}_{\max}$ is finite because (a) each element of $\mathbf{v}_{\max}$ is finite and (b) each element of $\mathbf{A}(Y_t)$ is finite. To see the latter, note that $\mathbf{z}_t$ and $\mathbf{x}_{t+1} - \mathbf{x}_t$ are bounded. As $\mathbf{A}(Y_t)\mathbf{v}_{\max}$ is finite, so is its norm $\|\mathbf{A}(Y_t)\mathbf{v}_{\max}\| = C_2$.

Thus, there exists constant $C = \max\{C_1, C_2\}$ such that the two conditions are true for all $t \geq 0$ and any starting conditions $\{S_0, \mathbf{z}_{-1}\}$. □

Lemma 4.5. *There exist constants $\rho \in (0, 1)$ and C such that for any initial conditions $Y_0 \in \mathcal{Y}$:*

1. $\|\mathbb{E}_d[\mathbf{b}(Y_t) \mid Y_0] - \mathbf{b}\| \leq C\rho^t$,

2. $\|\mathbb{E}_d[\mathbf{A}(Y_t) \mid Y_0] - \mathbf{A}\| \leq C\rho^t$.

Proof. 1. $\mathbf{b}(Y_t)$ in (4.11) is same as Tsitsiklis and Van Roy's (1999), so the first condition is true according to their Lemma 6.

2. $\mathbf{A}(Y_t)$ in (4.12) is slightly different than Tsitsiklis and Van Roy's (1999), so their Lemma 6 cannot be applied directly.

   Before we begin the proof, we state a property that shall be used repeatedly in this proof: for scalars $\gamma_1 < 1, \gamma_2 < 1, p, q$, there exist scalars $\gamma_3 < 1$ and r such that:

$$p\,\gamma_1^t + q\,\gamma_2^t \leq r\,\gamma_3^t \qquad \forall\ t \geq 0. \tag{4.15}$$

   Let us re-write the $\mathbf{A}(Y_t)$ matrix for both algorithms.

$$\mathbf{A}(Y_t)^{\text{Diff}} \doteq \begin{bmatrix} -\eta & \eta(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \\ -\mathbf{z}_t & \mathbf{z}_t(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \end{bmatrix} \qquad \text{from (4.12)}$$

$$\mathbf{A}(Y_t)^{\text{Avg}} \doteq \begin{bmatrix} -\eta & \mathbf{0}^\top \\ -\mathbf{z}_t & \mathbf{z}_t(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \end{bmatrix} \qquad \text{from Tsitsiklis \& Van Roy's (5)}$$

   Note that $\mathbf{A}(Y_t)^{\text{Diff}} = \mathbf{A}(Y_t)^{\text{Avg}} + \mathbf{F}_t$, where $\mathbf{F}_t = \begin{bmatrix} 0 & \eta(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \\ \mathbf{0} & \mathbf{0}\mathbf{0}^\top \end{bmatrix}$. Now,

$$\mathbb{E}[\mathbf{A}(Y_t)^{\text{Diff}} \mid Y_0] = \mathbb{E}[\mathbf{A}(Y_t)^{\text{Avg}} \mid Y_0] + \mathbb{E}[\mathbf{F}_t \mid Y_0]$$

$$\mathbb{E}[\mathbf{A}(Y_t)^{\text{Diff}} \mid Y_0] - \mathbf{A} = \mathbb{E}[\mathbf{A}(Y_t)^{\text{Avg}} \mid Y_0] - \mathbf{A} + \mathbb{E}[\mathbf{F}_t \mid Y_0]$$

$$\left\|\mathbb{E}[\mathbf{A}(Y_t)^{\text{Diff}} \mid Y_0] - \mathbf{A}\right\| \leq \left\|\mathbb{E}[\mathbf{A}(Y_t)^{\text{Avg}} \mid Y_0] - \mathbf{A}\right\| + \left\|\mathbb{E}[\mathbf{F}_t \mid Y_0]\right\|$$

   Tsitsiklis and Van Roy's (1999) Lemma 6 proves that there exist scalars C_0 and ρ_0 such that $\left\|\mathbb{E}[\mathbf{A}(Y_t)^{\text{Avg}}|S_0] - \mathbf{A}\right\| \leq C_0\rho_0^t$ for any $t \geq 0$ and any $S_0 \in \mathcal{S}$. Hence,

$$\left\|\mathbb{E}[\mathbf{A}(Y_t)^{\text{Diff}} \mid Y_0] - \mathbf{A}\right\| \leq C_0\rho_0^t + \left\|\mathbb{E}[\mathbf{F}_t \mid Y_0]\right\|. \tag{4.16}$$

   Now consider the second term on the RHS. In particular, consider its non-zero part: $\left\|\mathbb{E}[\eta(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \mid Y_0]\right\|$. Let $\mathbf{d}_\pi^t(i) = \Pr(S_t = s_i|S_0, \pi)$. Then,

$$\mathbb{E}[\eta(\mathbf{x}_{t+1} - \mathbf{x}_t) \mid Y_0] = \eta\mathbf{X}^\top(\mathbf{d}_\pi^{t+1} - \mathbf{d}_\pi^t).$$

Now, $\|\mathbf{d}_\pi^{t+1} - \mathbf{d}_\pi^t\| \leq \|\mathbf{d}_\pi^{t+1} - \mathbf{d}_\pi\| + \|\mathbf{d}_\pi^t - \mathbf{d}_\pi\|$. A well-known result for ergodic Markov chains is $\|\mathbf{d}_\pi^t - \mathbf{d}_\pi\| \leq C_1\rho_1^t$ for scalars $\rho_1 \in (0,1)$ and C_1 (e.g., Levin & Peres, 2017: Theorem 4.9). So $\|\mathbf{d}_\pi^{t+1} - \mathbf{d}_\pi^t\| \leq C_2\rho_2^t$ for scalars $\rho_2 < 1$ and C_2. Using that,

$$\left\|\mathbf{X}^\top(\mathbf{d}_\pi^{t+1} - \mathbf{d}_\pi^t)\right\| \leq \|\mathbf{X}\| \left\|\mathbf{d}_\pi^{t+1} - \mathbf{d}_\pi^t\right\| \leq K \cdot C_2\rho_2^t.$$

The final inequality follows from the property $\|\mathbf{Qr}\| \leq \|\mathbf{Q}\| \|\mathbf{r}\|$, and $\|\mathbf{X}\|$ is less than or equal to some constant K as all elements of $\mathbf{X}$ are bounded. Finally, $\|\mathbb{E}[\mathbf{F}_t \mid Y_0]\| = \|\eta(\mathbf{x}_{t+1} - \mathbf{x}_t)\|$. Substituting this in (4.16) and using (4.15), we get:

$$\left\|\mathbb{E}[\mathbf{A}(Y_t)^{\text{Diff}} \mid Y_0] - \mathbf{A}\right\| = \|\mathbb{E}[\mathbf{A}(Y_t) \mid Y_0] - \mathbf{A}\| \leq C\rho^t,$$

for some scalars $\rho \in (0,1)$ and C, for any $t \geq 0$ and any $Y_0 \in \mathcal{Y}$.

The proof is complete. □

We are now ready to state Tsitsiklis and Van Roy's (1999) stochastic-approximation result to show the convergence of Algorithm 1.

Theorem 4.2. *(Based on Tsitsiklis and Van Roy's (1999) Theorem 2 and Corollary 1) Consider the iterative algorithm of the form* $\mathbf{u}_{t+1} \doteq \mathbf{u}_t + \alpha_t(\mathbf{b}(Y_t) + \mathbf{A}(Y_t)\mathbf{u}_t)$. *Suppose the following conditions are satisfied:*

1. *The Markov chain* $\{Y_t\}$ *evolving in a state space* $\mathcal{Y}$ *has a unique steady-state distribution. Let* $\mathbb{E}_d[\cdot]$ *denote the expectation according to this distribution.*
2. *Let* $\mathbf{A} \doteq \mathbb{E}_d[\mathbf{A}(Y_t)]$. *There exists a diagonal matrix* $\mathbf{L}$ *with positive diagonal entries such that* $\mathbf{LA}$ *is negative definite.*
3. *There exists a constant* C *such that* $\|\mathbf{A}(Y)\| \leq C$ *and* $\|\mathbf{b}(Y)\| \leq C$ *for any* $Y \in \mathcal{Y}$.

4. *There exist scalars* C *and* $\rho \in (0,1)$ *such that* $\forall\, t \geq 0$ *and* $Y_0 \in \mathcal{Y}$*:*

$$\|\mathbb{E}[\mathbf{A}(Y_t) \mid Y_0] - \mathbf{A}]\| \leq C\rho^t,$$
$$\|\mathbb{E}[\mathbf{b}(Y_t) \mid Y_0] - \mathbf{b}]\| \leq C\rho^t, \quad \textit{where, } \mathbf{b} \doteq \mathbb{E}_d[\mathbf{b}(Y_t)].$$

5. *The step sizes* α_t *are positive, deterministic, and satisfy* $\sum_{t=0}^{\infty} \alpha_t = \infty$ *and* $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$.

Then $\mathbf{u}_t$ *converges to* $\mathbf{u}^*$ *with probability one, where* $\mathbf{u}^*$ *is the unique vector satisfying* $\mathbf{A}\mathbf{u}^* + \mathbf{b} = 0$.

We now verify that Algorithm 1 satisfies all the conditions of Theorem 4.2. As mentioned earlier, Condition 1 is satisfied by Yu's (2012, 2017) general analysis which is applicable for our joint chain $\{Y_t\}$ here. Lemma 4.3, 4.4, 4.5 establish that conditions 2, 3, and 4 are satisfied. Condition 5 is satisfied by Assumption 4.3.

Hence, Theorem 4.2 applies to Algorithm 1 and it converges to the unique fixed point $\mathbf{u}^*$ that satisfies $\mathbf{A}\mathbf{u}^* = \mathbf{b}$.

Characterizing the fixed point

We now characterize the fixed point $\mathbf{u}^*$. Let us consider the reward-rate scalar and the weight vector separately. Recall that the first term of the combined parameter vector $\mathbf{u} \in \mathbb{R}^{d+1}$ is the scalar reward-rate estimate and the rest is the weight vector.

From Lemma 4.2, $\mathbf{b}_0 = \eta\, r(\pi)$ and $\mathbf{A}_0 = [-\eta, \mathbf{0}^\top]^\top$. Hence, $\mathbf{u}_0^* = r(\pi)$. This proves part (1) of Theorem 4.1.

Proving part (2) takes a few more steps. Let us call $\mathbf{u}_{1:d}^* = \mathbf{w}^*$. Solving for $\mathbf{A}_{1:d}\mathbf{w}^* + \mathbf{b}_{1:d} = \mathbf{0}$, we have:

$$\underbrace{\frac{r(\pi)}{1-\lambda}\mathbf{X}^\top \mathbf{D}_\pi \mathbf{1}}_{\text{Term 1}} + \underbrace{\sum_{k=0}^{\infty} \lambda^k \mathbf{X}^\top \mathbf{D}_\pi \mathbf{P}_\pi^k (\mathbb{I} - \mathbf{P}_\pi)\mathbf{X}\mathbf{w}^*}_{\text{Term 2}} = \underbrace{\sum_{k=0}^{\infty} \lambda^k \mathbf{X}^\top \mathbf{D}_\pi \mathbf{P}_\pi^k \mathbf{r}_\pi}_{\text{Term 3}}. \tag{4.17}$$

Let us consider the second term. Recall from (4.2) that we defined $\mathbf{P}_\pi^\lambda \doteq (1-\lambda)\sum_{n=1}^\infty \lambda^{n-1}\mathbf{P}_\pi^n$. Now,

$$\begin{aligned}\sum_{k=0}^\infty \lambda^k \mathbf{P}_\pi^k(\mathbb{I}-\mathbf{P}_\pi) &= (\mathbb{I}-\mathbf{P}_\pi)+(\lambda\mathbf{P}_\pi-\lambda\mathbf{P}_\pi^2)+(\lambda^2\mathbf{P}_\pi^2-\lambda^2\mathbf{P}_\pi^3)+\dots\\ &= \mathbb{I}+\frac{\lambda}{1-\lambda}\mathbf{P}_\pi^\lambda-\frac{1}{1-\lambda}\mathbf{P}_\pi^\lambda\\ &= \mathbb{I}-\mathbf{P}_\pi^\lambda.\end{aligned}$$

Thus, the second term can be written as $\mathbf{X}^\top\mathbf{D}_\pi(\mathbb{I}-\mathbf{P}_\pi^\lambda)\mathbf{X}\mathbf{w}^*$. Substituting this in (4.17),

$$\frac{r(\pi)}{1-\lambda}\mathbf{X}^\top\mathbf{D}_\pi\mathbf{1}+\mathbf{X}^\top\mathbf{D}_\pi(\mathbb{I}-\mathbf{P}_\pi^\lambda)\mathbf{X}\mathbf{w}^*=\sum_{k=0}^\infty\lambda^k\mathbf{X}^\top\mathbf{D}_\pi\mathbf{P}_\pi^k\mathbf{r}_\pi.$$

On re-arranging, we get:

$$\begin{aligned}\mathbf{X}^\top\mathbf{D}_\pi\mathbf{X}\mathbf{w}^* &= \sum_{k=0}^\infty\lambda^k\mathbf{X}^\top\mathbf{D}_\pi\mathbf{P}_\pi^k\mathbf{r}_\pi-\frac{r(\pi)}{1-\lambda}\mathbf{X}^\top\mathbf{D}_\pi\mathbf{1}+\mathbf{X}^\top\mathbf{D}_\pi\mathbf{P}_\pi^\lambda\mathbf{X}\mathbf{w}^*\\ &= \mathbf{X}^\top\mathbf{D}_\pi\Big(\sum_{k=0}^\infty\lambda^k\mathbf{P}_\pi^k\mathbf{r}_\pi-\frac{r(\pi)}{1-\lambda}\mathbf{1}+\mathbf{P}_\pi^\lambda\mathbf{X}\mathbf{w}^*\Big)\\ &= \mathbf{X}^\top\mathbf{D}_\pi T_\pi^\lambda(\mathbf{X}\mathbf{w}^*)\qquad\text{(from (4.3))}\end{aligned}$$

Multiplying both sides by $\mathbf{X}(\mathbf{X}^\top\mathbf{D}_\pi\mathbf{X})^{-1}$,

$$\begin{aligned}\mathbf{X}\mathbf{w}^* &= \mathbf{X}(\mathbf{X}^\top\mathbf{D}_\pi\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{D}_\pi T_\pi^\lambda(\mathbf{X}\mathbf{w}^*)\\ &= \Pi T_\pi^\lambda(\mathbf{X}\mathbf{w}^*)\end{aligned}$$

This shows part (2) of Theorem 4.1.

We have shown parts 1 and 2 of Theorem 4.1, that Algorithm 1 converges with probability one: $\bar{R}$ to $r(\pi)$, and $\mathbf{w}$ to a solution of $\mathbf{X}\mathbf{w}=\Pi\,T^\lambda\big(\mathbf{X}\mathbf{w}\big)$.

Finally, the error bound is same as Tsitsiklis and Van Roy's (1999) Lemma 3 because the fixed points of on-policy linear Differential TD(λ) and Average-Cost TD(λ) are the same. □

4.3 On-policy: Experiments

In this section, we present experiments to test the efficacy of the proposed on-policy Differential TD(λ) algorithm. In particular, we would like to validate that the algorithm indeed converges to the fixed point predicted by the theory. We would also like to validate a straightforward hypothesis from the discounted-reward formulation: an intermediate value of λ typically leads to the faster rate of learning compared to values close to either extreme. Finally, it is pertinent to compare the performance of Differential TD(λ) with Average-Cost TD(λ) since they are both algorithms for the on-policy average-reward prediction problem.

We ran a set of experiments on a random-walk domain. Consider an MDP with N states and two actions—left and right—available in each state. The states are arranged in a chain such that the left and right actions lead deterministically to the previous and next states respectively (see Figure 4.1). The right action in the right-most state and the left action in the left-most state lead to the middle-most state with a reward of $+1$ and -1 respectively. All the other transitions result in zero reward. This problem is a continuing variant of the episodic random-walk problem used often by Sutton and Barto (2018) to demonstrate the effects of n-step and multi-step methods in the episodic setting.

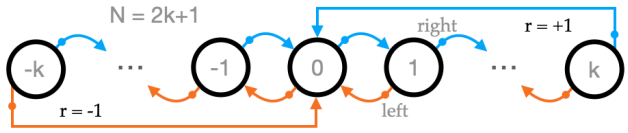


Figure 4.1: A continuing N-state random-walk problem.

For this on-policy experiment, we evaluated a policy that takes the right and left actions in each of the $N = 19$ states with equal probability. The average reward corresponding to this policy is zero. The agent observed a one-hot representation of the state (a vector of the size of the state space with all zeros except 1 at the position of the current state), which made it easy to evaluate convergence as the differential

value function was completely representable. We applied Differential TD(λ) and Average-Cost TD(λ) to this problem with $\alpha \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}, \eta \in \{0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0\}$, and $\lambda \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99\}$. Each parameter configuration was run for 10,000 steps and repeated 100 times. The step sizes were decayed by a factor of 0.99975 at each time step. The weight vector, trace vector, and the average-reward estimate were initialized to zero.

As with the one-step prediction experiments in Chapter 3, we evaluated the quality of the reward-rate estimate by the squared error w.r.t. the true reward rate of the target policy; for value estimates, we computed the root mean squared error of the estimated value function w.r.t. the nearest solution of the Bellman equations, which we also referred to as 'RMSVE (TVR)' in Chapter 3.

Figure 4.2 shows some learning curves corresponding to different values of λ on the 19-state random-walk problem. Each point on the solid lines represent the mean error across the 100 runs for a particular time step; the shaded region represents one standard error. The errors are not smoothed or binned. The learning curves for each λ correspond to the step size α that resulted in the fastest learning (measured by the area under the RMSVE curve) for a fixed value of $\eta = 0.03$; the trends were similar

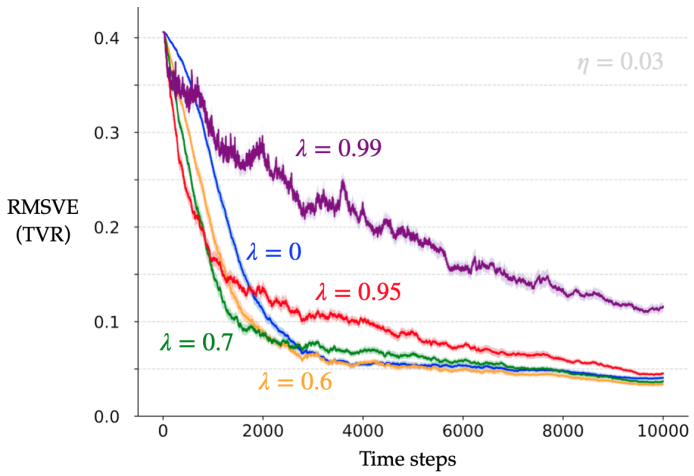


Figure 4.2: Learning curves corresponding to different values of λ on the 19-state random-walk problem using Algorithm 1.

for other values of η.

We saw that an intermediate value of λ resulted in fast learning as well as low asymptotic error. The learning curves all approached zero after about 30k steps, except for $\lambda = 0.99$, which took longer; only 10k steps are shown here to focus on the differences in the early part of learning. Note that in all cases the average-reward estimate converged to the true average reward of zero; however, those learning curves are uninteresting given the estimates are initialized to zero and hence are omitted here. For all the multi-step methods, the value-error-reduction rate in the beginning was higher than that of the one-step method ($\lambda = 0$). However, the rate of learning was much lower for larger values of λ because of increasing variance (which is reflected in the roughness of curves).

The sensitivity plots in Figure 4.3 further corroborate these trends. Each point denotes the value error averaged over the first 2000 steps of training (to reflect the initial rate of learning) and the error bars represent one standard error over the 100 independent runs of each parameter setting. From the plot on the left corresponding to Algorithm 1, we saw that $\lambda = 0$ resulted in slow learning across the range of step size α tested. Large values of λ resulted in slow learning or numerical divergence of values due to large magnitudes of the trace parameter. Intermediate values of λ

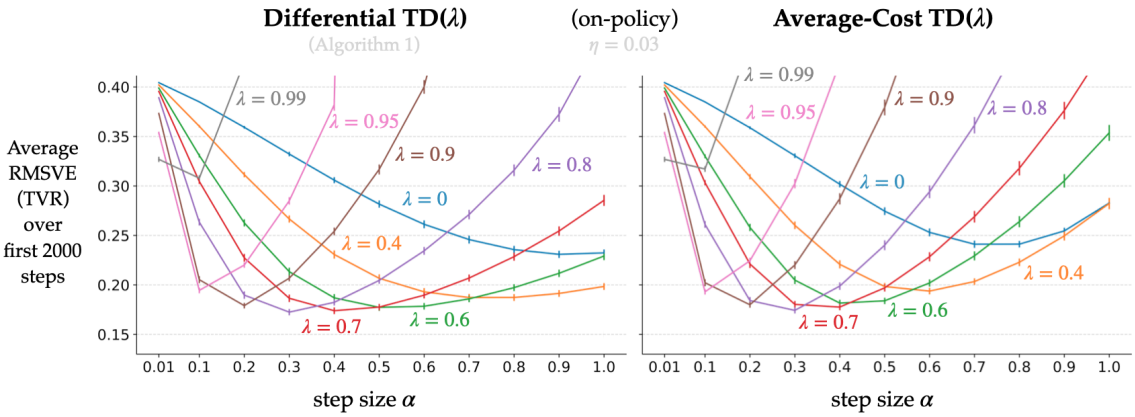


Figure 4.3: Parameter studies showing the sensitivity of the two algorithms' performance to their parameters α and λ.

resulted in the fastest learning, but for different values of α. The sensitivity plots in Figure 4.3 correspond to $\eta = 0.03$; the trends were similar for other values of η.

Figure 4.3 also helped us compare the performance of Algorithm 1 with that of Average-Cost TD(λ) in the on-policy setting. We found that the performance of the two on-policy algorithms was quite similar, with Average-Cost TD(λ) slightly more sensitive to large values of α. Figure 4.4 shows another slice of the comparison between the two, now for a fixed value of $\lambda = 0.7$. The y-axis corresponds to the value error averaged over the entire training period. Both algorithms behaved similarly across values of α, however, Average-Cost TD(λ) was less sensitive to larger values of η. This is likely because the average-reward update for Average-Cost TD(λ) does not depend on the values and hence can tolerate a larger step size relative to the value updates. The trends were not very different for other values of λ.

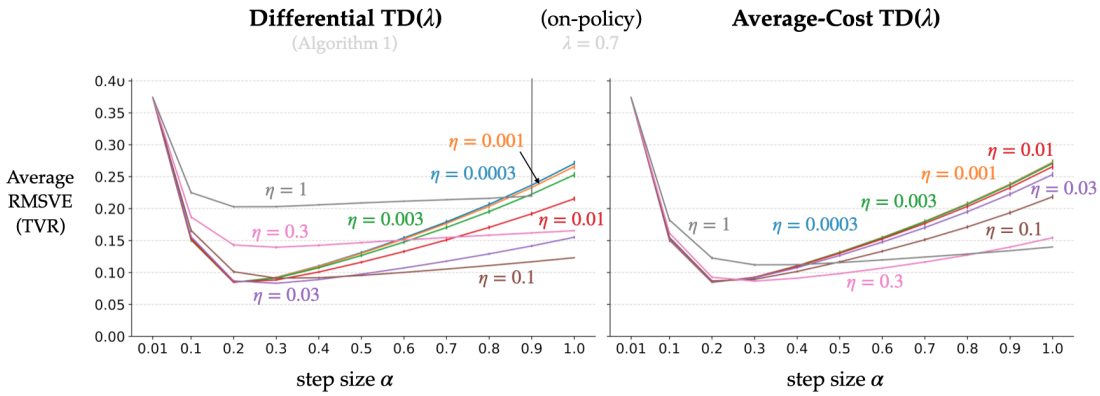


Figure 4.4: Parameter studies showing the sensitivity of the two algorithms' performance to their parameters α and η.

From these simple experiments we conclude that Algorithm 1 works as intended: in the tabular case, the value and average-reward estimates converge to a solution of the differential Bellman equations (2.6). Further, an intermediate value of λ can be a sweet spot in trading off the speed of learning and the variance in updates. The performance of Algorithm 1 was comparable to Average-Cost TD(λ). This was good to know because the differential style of TD-error update was adapted for the off-policy setting, which I investigate next.

4.4 Off-policy: An Unsuccessful Attempt

In Section 4.2 we proved the convergence of Algorithm 1, which is a straightforward extension of one-step tabular Differential TD-learning from Chapter 3. Algorithm 1 and Average-Cost TD(λ) are two multi-step prediction algorithms that are proved to converge in the on-policy setting with linear function approximation. However, the main reason for extending the one-step differential algorithms to the multi-step case was to obtain the first multi-step average-reward algorithm that is convergent in the *off-policy* setting. Unfortunately, Algorithm 1 is restricted to the on-policy setting; it cannot be extended to the off-policy setting even in the tabular case.

Note that our aim for the off-policy setting is the simplest tabular case. Off-policy RL with general function approximation is an open problem even in the well-studied discounted-reward formulation. Even the special case of linear function approximation is unresolved. The main challenge of off-policy learning with function approximation comes from two kinds of mismatches: (1) the mismatch in the state distribution as a result of following the behavior policy instead of the target policy, and (2) the mismatch in the rewards obtained due to the differences in action-selection probabilities in a given state. In the tabular discounted case, the former mismatch is not a concern because the value of each state is estimated independently, and using importance-sampling ratios appropriately corrects the latter mismatch and leads to an unbiased estimate of the values in each state (Rubinstein, 1981; Precup et al., 2000). However, with general function approximation, the values of different states depend on each other and hence addressing the mismatch in state distributions is relevant. This is an active area of research.

The methods developed in the last few years either try to correct the state-distribution mismatch (e.g., Hallak & Mannor, 2017; Gelada & Bellemare, 2019; Zhang et al., 2020b) or embrace the mismatch in favor of a different objective (e.g., Maei, 2011; Hackman, 2012; White & White, 2016; Sutton et al., 2016). Each of these

methods involves significant conceptual and algorithmic complexity. In this work, we focus on the simpler tabular case; extensions to general function approximation for the off-policy setting would be a complete dissertation in itself.

While we consider the extension of Algorithm 1 to the tabular off-policy setting, we continue using the linear terminology and notation for the sake of brevity. The observation vectors are now a one-hot representation of the index of each state. That is, if each state $s \in \mathcal{S}$ is identified by a number from 0 to $|\mathcal{S}|-1$, then the observation vector corresponding to the i-th state S_i is the vector $\mathbf{x}(s_i) = \mathbf{1}_i$, which is a $|\mathcal{S}|$-dimensional vector with a 1 at the i-th index and zeros everywhere else. The state-value estimate of the i-th state is then just the value of the i-th index of the weight vector $\mathbf{w} \in \mathbb{R}^{|\mathcal{S}|}$: $\hat{v}(s_i) \doteq \mathbf{w}^\top \mathbf{x}(s_i) = \mathbf{w}^\top \mathbf{1}_i = w_i$.

Recall that the tabular version of Algorithm 1 updates the weight vector and the reward-rate estimate after each transition $(S_t, A_t, R_{t+1}, S_{t+1})$ as:

$$\begin{aligned}
\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha_t \delta_t \mathbf{z}_t, \\
\bar{R}_{t+1} &\doteq \bar{R}_t + \eta \alpha_t \delta_t, \\
\text{where,} \quad \mathbf{z}_t &\doteq \lambda \mathbf{z}_{t-1} + \mathbf{x}(S_t), \\
\text{and,} \quad \delta_t &\doteq R_{t+1} - \bar{R}_t + \mathbf{w}_t^\top \mathbf{x}(S_{t+1}) - \mathbf{w}_t^\top \mathbf{x}(S_t).
\end{aligned}$$

To extend this algorithm to the tabular off-policy setting, we just have to correct the mismatch of targets due to the difference in actions taken by the target and behavior policies. So the straightforward extension of Algorithm 1 to the tabular off-policy case—let us call it *Algorithm 1-off*—updates its parameters as:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha_t \delta_t \mathbf{z}_t, \tag{4.18}$$

$$\bar{R}_{t+1} \doteq \bar{R}_t + \eta \alpha_t \rho_t \delta_t, \tag{4.19}$$

$$\text{where,} \quad \mathbf{z}_t \doteq \rho_t (\lambda \mathbf{z}_{t-1} + \mathbf{x}(S_t)), \tag{4.20}$$

$$\begin{aligned}
\delta_t &\doteq R_{t+1} - \bar{R}_t + \mathbf{w}_t^\top \mathbf{x}(S_{t+1}) - \mathbf{w}_t^\top \mathbf{x}(S_t), \\
\text{and,} \quad \rho_t &\doteq \frac{\pi(A_t|S_t)}{b(A_t|S_t)}.
\end{aligned}$$

This algorithm looks promising. One might rightly point out that the trace parameter updated in this way is unbounded in general (see, e.g., Yu's (2012) Proposition 3.1). However, there is a deeper issue even after we bound the trace parameter (by, say, limiting the choice of λ such that $\lambda\rho_{\max} < 1$, where $\rho_{\max}$ is the largest importance-sampling ratio given the target and behavior policies).

The issue is that this algorithm is not stable. That is, for certain problems, the algorithm will diverge from any initialization. This is not obvious. It took me several months of attempting to prove the convergence of this algorithm before realizing it is impossible. In particular, I showed that this algorithm satisfies all the conditions of Theorem 4.2 that was used to prove the convergence of Algorithm 1 (and Tsitsiklis and Van Roy's Average-Cost TD(λ)) *except* that $\mathbf{A}$ or the diagonally scaled version of $\mathbf{A}$ is negative definite.

However, negative definiteness is a sufficient and not a necessary condition for for stability. The necessary condition is that the real parts of all the eigenvalues of a matrix should be negative for stability—such a matrix is also called a *Hurwitz* matrix. So despite being unable to prove that $\mathbf{A}$ was negative definite, I had hope. At that time, I was attributing my lack of success to a lack of formal training in theory. But I realized I could numerically test the hypothesis that $\mathbf{A}$ is Hurwitz.

For the tabular off-policy version of Algorithm 1, I derived:

$$\mathbf{A} = \begin{bmatrix} -\eta & \eta\mathbf{d}_b^\top(\mathbf{P}_\pi - \mathbb{I}) \\ \frac{-1}{1-\lambda}\mathbf{D}_b\mathbf{1} & \sum_{k=0}^{\infty}\lambda^k\mathbf{D}_b\mathbf{P}_\pi^k(\mathbf{P}_\pi - \mathbb{I}) \end{bmatrix}_{(d+1)\times(d+1)} \tag{4.21}$$

One way to test if $\mathbf{A}$ is Hurwitz or not is to generate arbitrary $\mathbf{d}_b$ and $\mathbf{P}_\pi$ and check the eigenvalues of the resulting $\mathbf{A}$ matrices for different values of η and λ. This is easy with standard libraries such as `numpy`. Indeed, I found that for many combinations of randomly generated state distribution vectors $\mathbf{d}_b$ and stochastic matrices $\mathbf{P}_\pi$, the matrix $\mathbf{A}$ has eigenvectors with positive real parts. In other words, there exist off-policy prediction problems for which the simple extension of Algorithm 1 to the off-policy setting diverges for any initial conditions even in the tabular case.

As a result, I had to go back to the drawing board in the quest for a multi-step average-reward prediction off-policy algorithm for the tabular case.

4.5 Off-policy: Theory

The previous failure was puzzling. The off-policy problem is hard, but its tabular version should be relatively straightforward. Precup, Sutton, and Singh (2000) showed that in the discounted reward setting, the tabular multi-step algorithm is the obvious extension of the one-step algorithm using eligibility traces. There did not appear to be an obvious reason why the average-reward variant would be more complicated. What could have been the issue?

It turned out that Algorithm 1 was not the most faithful extension of one-step Differential TD-learning to the multi-step setting with eligibility traces. By creating a version that was, we showed the convergence of a multi-step off-policy tabular average-reward prediction algorithm. I now discuss in detail what the shortcoming of Algorithm 1 was and how we fixed it.

Recall the updates of *one-step* off-policy Differential TD-learning, now expressed in terms of vector updates:

$$\begin{aligned}
\mathbf{v}_{t+1} &\doteq \mathbf{v}_t + \alpha_t \rho_t \delta_t \mathbf{1}_{S_t}, \\
\bar{R}_{t+1} &\doteq \bar{R}_t + \eta \alpha_t \rho_t \delta_t, \\
\text{where,} \quad \delta_t &\doteq R_{t+1} - \bar{R}_t + \mathbf{v}_t(S_{t+1}) - \mathbf{v}_t(S_t), \\
\text{and,} \quad \rho_t &\doteq \frac{\pi(A_t|S_t)}{b(A_t|S_t)},
\end{aligned}$$

where $\mathbf{1}_{S_t}$ is a vector of all zeros with 1 at the index of the S_t-th state. Starting from arbitrary initial values of the value estimates—$\mathbf{v}_0$—and the reward-rate estimate—$\bar{R}_0$, the following relation is preserved for all $t > 0$:

$$\begin{aligned}
\bar{R}_t - \bar{R}_0 &= \eta \Big[\sum_s \mathbf{v}_t(s) - \sum_s \mathbf{v}_0(s) \Big], \\
\text{or,} \quad \bar{R}_t - \bar{R}_0 &= \eta \mathbf{1}^\top (\mathbf{v}_t - \mathbf{v}_0). \qquad (4.22)
\end{aligned}$$

This is because both the reward-rate estimate and the value estimates are updated by the same quantity, so the total change in both sets of estimates at any point of time is the same (scaled by η). This is important because the reward rate can then be estimated as a linear or affine function of the value estimates. Such an implicit estimation enables using the techniques that are used to analyze the relative-value-iteration-style (RVI) algorithms. In particular, Abounadi et al. (2001) analyzed this self-referential mechanism of estimating the reward rate for their tabular one-step RVI Q-learning algorithm (discussed in Chapter 3) using Borkar's (2009) ODE approach. Yi Wan then generalized such an analysis for one-step Differential Q-learning in our paper (Wan, Naik, & Sutton, 2021a).

Without loss of generality, we can assume that the average-reward estimate and the value estimates to be initialized to zero, in which case the relation in (4.22) simplifies to $\bar{R}_t = \eta \mathbf{1}^\top \mathbf{v}_t$. Now, note that Algorithm 1-off does not maintain this relation because, for $\lambda > 0$, the updates to the values estimates have more components due to the trace:

$$\mathbf{v}_{t+1} \doteq \mathbf{v}_t + \alpha_t \delta_t \mathbf{z}_t,$$
$$\bar{R}_{t+1} \doteq \bar{R}_t + \eta \alpha_t \rho_t \delta_t,$$

where $\mathbf{z}_t \doteq \rho_t(\lambda \mathbf{z}_{t-1} + \mathbf{1}_{S_t})$ and $\delta_t \doteq R_{t+1} - \bar{R}_t + \mathbf{v}_t(S_{t+1}) - \mathbf{v}_t(S_t)$.

We can build upon the body of RVI-style methods if the average-reward estimate is a function of the value estimates. One way to ensure (4.22) still holds is to have a trace parameter *for the reward-rate estimate*, which is just a scalar trace of 1.

Tabular off-policy Differential TD(λ) (Algorithm 2)

In addition to the usual trace vector $\mathbf{z} \in \mathbb{R}^{|\mathcal{S}|}$, consider a scalar $z^{\bar{R}} \in \mathbb{R}$. After observing S_t, taking A_t according to b, observing S_{t+1} and R_{t+1}:

$$\mathbf{v}_{t+1} \doteq \mathbf{v}_t + \alpha_t \delta_t \mathbf{z}_t, \tag{4.23}$$

$$\bar{R}_{t+1} \doteq \bar{R}_t + \eta \alpha_t \delta_t z_t^{\bar{R}}, \tag{4.24}$$

$$\text{where,} \quad \mathbf{z}_t \doteq \rho_t(\lambda \mathbf{z}_{t-1} + \mathbf{1}_{S_t}), \tag{4.25}$$

$$z_t^{\bar{R}} \doteq \rho_t(\lambda z_{t-1}^{\bar{R}} + 1), \tag{4.26}$$

with the usual $\delta_t \doteq R_{t+1} - \bar{R}_t + \mathbf{v}_t(S_{t+1}) - \mathbf{v}_t(S_t)$ and $\rho_t \doteq \dfrac{\pi(A_t|S_t)}{b(A_t|S_t)}$.

Note that in the on-policy setting, $z^{\bar{R}}$ saturates to $1/(1-\lambda)$, which can be interpreted as a scaling of the step size which makes it equivalent to Algorithm 1 (and further to on-policy Differential TD(0) from Chapter 3 when $\lambda = 0$). However, in the off-policy case, it is different from Algorithm 1-off in that it ensures (4.22) is satisfied for all $t > 0$.

More generally, there is a larger family of algorithms that maintains $\bar{R}_t = f(\mathbf{v}_t)$, that is, where the reward-rate estimate is a function $f : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}$ of the current values estimates:

$$\mathbf{v}_{t+1} \doteq \mathbf{v}_t + \alpha_t \big[R_{t+1} - f(\mathbf{v}_t) + \mathbf{v}_t(S_{t+1}) - \mathbf{v}_t(S_t)\big]\mathbf{z}_t, \tag{4.27}$$

$$\text{where,} \quad \mathbf{z}_t \doteq \rho_t(\lambda \mathbf{z}_{t-1} + \mathbf{1}_{S_t}), \tag{4.28}$$

$$\text{and,} \quad \rho_t \doteq \pi(A_t|S_t)/b(A_t|S_t). \tag{4.29}$$

We prove the stability and convergence of a general family of algorithms where f is of the form $f(\mathbf{v}_t) = \eta\, \mathbf{g}^\top \mathbf{v}_t$, with $\eta > 0$ and $\mathbf{g} \in \mathbb{R}^{|\mathcal{S}|}$ according to the following assumption.

Convergence Theorem

Assumption 4.4. $\mathbf{g} \in \mathbb{R}^{|\mathcal{S}|}$ *is a non-negative vector with at least one positive element.*

Note that Algorithm 2 is a specific instance of this family with $\mathbf{g} = \mathbf{1}$.

Assumption 4.5. *The behavior policy has coverage, that is,* $b(a|s) > 0$ *wherever* $\pi(a|s) > 0$, $\forall s, a$.

This assumption—often referred to as the *coverage* assumption in the off-policy literature—guarantees that all the states that occur under the target policy also occur under the behavior policy b. Let $\mathbf{d}_b$ denote unique steady-state distribution over the states in the Markov chain induced by the behavior policy.

The trace parameter $\mathbf{z}_t$ can in general become unbounded in the off-policy setting for arbitrary target and behavior policies (see, e.g., Yu's (2012) Proposition 3.1). Our proof technique requires boundedness for all $t > 0$, which we impose with a simple condition.

Assumption 4.6. $\lambda\rho_{\max} < 1$, *where* $\rho_{\max}$ *is the largest importance-sampling ratio.*

Theorem 4.3. *Under Assumptions 4.3–4.6, for a given* $\lambda > 0$, *there exists a sufficiently small* η *for which the family of tabular off-policy Differential TD(*λ*) algorithms (4.27–4.29) converges almost surely to a solution* $\mathbf{v}_\infty$ *of the differential Bellman equations* (4.3), *with* $f(\mathbf{v}_\infty) = r(\pi)$, *where* $r(\pi)$ *denotes the reward rate of the target policy* π.

Convergence Proof

We use the ordinary-differential-equation (ODE) approach in the style of Borkar (2009) to prove the stability and convergence of this family of algorithms. In particular, we show the sequence of estimates generated by the algorithm is (1) bounded and (2) asymptotically converges to the solutions of an ODE. We further show that the key matrix involved in the analysis is Hurwitz, which implies that the ODE has a stable unique equilibrium point and that the algorithm converges to it asymptotically.

Preliminaries

We begin by casting the algorithmic updates in the form of a general stochastic-approximation algorithm. To (4.27), add and subtract $r(S_t, A_t, S_{t+1})\mathbf{z}_t$, where, $r(S_t, A_t, S_{t+1}) \doteq \sum_r p(r \mid S_t, A_t, S_{t+1})\, r$,

$$\begin{aligned}\mathbf{v}_{t+1} &= \mathbf{v}_t + \alpha_t\, \mathbf{z}_t\big(r(S_t, A_t, S_{t+1}) - \eta\, \mathbf{g}^\top \mathbf{v}_t + v_t(S_{t+1}) - v_t(S_t) \\ &\qquad + R_{t+1} - r(S_t, A_t, S_{t+1})\big) \\ &= \mathbf{v}_t + \alpha_t\big[h(\mathbf{v}_t, Y_t) + \mathbf{m}_{t+1}\big], \end{aligned} \tag{4.30}$$

where $Y_t \doteq (\mathbf{z}_t, S_t, A_t, S_{t+1})$,

$$\mathbf{m}_{t+1} \doteq \big(R_{t+1} - r(S_t, A_t, S_{t+1})\big)\mathbf{z}_t, \tag{4.31}$$

$$h(\mathbf{v}_t, Y_t) \doteq \mathbf{b}_t + \mathbf{A}_t \mathbf{v}_t, \tag{4.32}$$

$$\text{with,} \quad \mathbf{b}_t \doteq r(S_t, A_t, S_{t+1})\, \mathbf{z}_t, \tag{4.33}$$

$$\text{and,} \quad \mathbf{A}_t \doteq \mathbf{z}_t(\mathbf{1}_{S_{t+1}} - \mathbf{1}_{S_t} - \eta\, \mathbf{g})^\top. \tag{4.34}$$

Note that the joint chain $\{Y_t\}$ is Markov: the states evolve according to the transition dynamics of the original MRP, the actions are taken according to a Markov policy b, and $\mathbf{z}_t$ evolves as $\mathbf{z}_t = \rho_t(\lambda \mathbf{z}_{t-1} + \mathbf{x}_t)$, where $\rho_t = \pi(A_t|S_t)/b(A_t|S_t)$. Furthermore, Yu (2012, 2017) showed (for the more general case without bounded traces) that $\{Y_t\}$ is a weak Feller Markov chain that has a unique invariant probability measure d.

The stability and convergence of the general algorithm depends on the expected values of $\mathbf{A}_t$ and $\mathbf{b}_t$ (and hence $h(\mathbf{v}, Y_t)$) w.r.t. d. We first show that the matrix key to the analysis, $\mathbf{A} = \mathbb{E}_d[\mathbf{A}_t]$, is Hurwitz for sufficiently small η, which implies the ODE $\dot{\mathbf{u}} = \mathbf{A}\mathbf{u} + \mathbf{b}$ converges to a unique stable solution $\mathbf{u}^*$ of $\mathbf{A}\mathbf{u} + \mathbf{b} = 0$ from any initial condition (Khalil, 2002), where $\mathbf{b} = \mathbb{E}_d[\mathbf{b}_t]$. Then we show the properties of h and $\mathbf{m}$ such that asymptotic sequence of discrete iterates $\{\mathbf{v}_t\}$ generated by (4.30) is bounded and converges to the unique stable equilibrium point of the ODE (Borkar, 2009). At the end, we show that the equilibrium point $\mathbf{u}^*$ indeed satisfies the differential Bellman equations.

The corresponding ODE has a unique equilibrium point

To compute the expectations $\mathbf{A}$ and $\mathbf{b}$, we again consider a double-ended stationary Markov chain (like in the on-policy analysis). $\{Y_t\}$ evolving in $\mathcal{Y}$ such that for all integers $t \in (-\infty, \infty)$, $\Pr(\mathbf{z}_t \in \mathcal{B}, S_t = s, A_t = a, S_{t+1} = s') = d(\mathcal{B}, s, a, s')$, where $\mathcal{B}$ is any Borel subset of $\mathcal{Y}$. Let $\mathbb{E}_d$ also denote the expectations w.r.t. the double-ended stationary Markov chain.

We now derive some expressions of interest.

Lemma 4.6. *Under Assumptions 4.1 and 4.5,*

1. $\mathbb{E}_d[\mathbf{z}_t] = \frac{1}{1-\lambda}\mathbf{D}_b\mathbf{1}$,
2. $\mathbb{E}_d[\mathbf{z}_t\, r(S_t, A_t, S_{t+1})] = \sum_{k=0}^{\infty} \lambda^k \mathbf{D}_b \mathbf{P}_\pi^k \mathbf{r}_\pi$,
3. $\mathbb{E}_d\big[\mathbf{z}_t(\mathbf{1}_{S_{t+1}} - \mathbf{1}_{S_t})^\top\big] = \mathbf{D}_b(\mathbf{P}_\pi^\lambda - \mathbb{I})$.

Proof. First, note that the expressions are the same for any time step t because they are expectations of functions of Y under the invariant probability measure d. So, without loss of generality, we compute the expressions for $t = 0$. Now, based on Yu's (2012, 2017) analysis, we can write $\mathbf{z}_0 = \sum_{k=-\infty}^{0} \rho_{k:0}\lambda^{-k}\mathbf{x}_k$ almost surely, where $\rho_{k:0} \doteq \Pi_{i=k}^{0}\rho_i$ and $\mathbf{x}_k$ denotes the feature vector at the k-th time step. Next, note that the trace parameter does not affect the evolution of the original Markov chain $\{S_t\}$ induced by the behavior policy b, hence the marginal distribution of states in Y is the same as $\mathbf{d}_b$ from the original Markov chain.

1. We now compute $\mathbb{E}_d[\mathbf{z}_0]$.

$$\begin{aligned}\mathbb{E}_d[\mathbf{z}_0] &= \mathbb{E}_d\Big[\sum_{k=-\infty}^{0} \lambda^{-k}\rho_{k:0}\mathbf{x}_k\Big] \\ &= \sum_{k=-\infty}^{0} \lambda^{-k}\mathbb{E}_d[\rho_{k:0}\mathbf{x}_k]\end{aligned}$$

   Note that the form of $\mathbf{z}_0 = \sum_{k=-\infty}^{0} \rho_{k:0}\lambda^{-k}\mathbf{x}_k$ enables swapping the expectation and the summation and to compute the individual expectations of quantities of

finite lengths. In particular, for $k < 0$, $\mathbb{E}_d[\rho_{k:0}\mathbf{x}_k]$ simply is $\sum_{s_k} d_b(s_k)\Big(\mathbf{x}(s_k)\Big) = \mathbf{d}_b = \mathbf{D}_b\mathbf{1}$. Then,

$$\begin{aligned}\mathbb{E}_d[\mathbf{z}_0] &= \sum_{k=-\infty}^{0} \lambda^{-k}\mathbb{E}_d[\rho_{k:0}\mathbf{x}_k] \\ &= \sum_{k=-\infty}^{0} \lambda^{-k}\mathbf{D}_b\mathbf{1} = \sum_{k=0}^{\infty}\lambda^k\mathbf{D}_b\mathbf{1} \\ &= \frac{1}{1-\lambda}\mathbf{D}_b\mathbf{1}.\end{aligned}$$

2. We again compute the value at $t = 0$:

$$\begin{aligned}\mathbb{E}_d[\mathbf{z}_0\, r(S_0, A_0, S_1)] &= \mathbb{E}_d[\sum_{k=-\infty}^{0} \lambda^{-k}\rho_{k:0}\mathbf{x}_k\, r(S_0, A_0, S_1)] \\ &= \sum_{k=-\infty}^{0} \lambda^{-k}\mathbb{E}_d[\rho_{k:0}\mathbf{x}_k\, r(S_0, A_0, S_1)].\end{aligned}$$

Computing the general form of the individual terms:

$$\begin{aligned}\mathbb{E}_d[\rho_{k:0}\mathbf{x}_k\, r(S_0, A_0, S_1)] &= \sum_{s_k} d_b(s_k) \sum_{a_k} b(a_k|s_k) \sum_{s_{k+1}} p(s_{k+1}|s_k, a_k) \sum_{a_{k+1}} b(a_{k+1}|s_{k+1}) \dots \\ &\qquad \sum_{s_0} p(s_0|s_{-1}, a_{-1}) \sum_{a_0} b(a_0|s_0) \sum_{r} p(r|s_0, a_0) \\ &\qquad\qquad \Big(\frac{\pi(a_k|s_k)}{b(a_k|s_k)} \frac{\pi(a_{k+1}|s_{k+1})}{b(a_{k+1}|s_{k+1})} \dots \frac{\pi(a_0|s_0)}{b(a_0|s_0)}\, r\, \mathbf{x}(s_k)\Big) \\ &= \sum_{s_k} d_b(s_k) \sum_{a_k} \pi(a_k|s_k) \sum_{s_{k+1}} p(s_{k+1}|s_k, a_k) \sum_{a_{k+1}} \pi(a_{k+1}|s_{k+1}) \dots \\ &\qquad \sum_{s_0} p(s_0|s_{-1}, a_{-1}) \sum_{a_0} \pi(a_0|s_0) \sum_{r} p(r|s_0, a_0) \Big(r\, \mathbf{x}(s_k)\Big) \\ &= \sum_{s_k} d_b(s_k)\mathbf{x}(s_k) \sum_{s_0} \Pr(s_0|s_k, -k, \pi) r_\pi(s_0) \\ &= \mathbf{X}^\top \mathbf{D}_b \mathbf{P}_\pi^{-k} \mathbf{r}_\pi \\ &= \mathbf{D}_b \mathbf{P}_\pi^{-k} \mathbf{r}_\pi \qquad \text{(in the tabular case).}\end{aligned}$$

Now,

$$\begin{aligned}\mathbb{E}_d[\mathbf{z}_0\, r(S_0, A_0, S_1)] &= \sum_{k=-\infty}^{0} \lambda^{-k} \mathbb{E}_d[\rho_{k:0}\mathbf{x}_k\, r(S_0, A_0, S_1)] \\ &= \sum_{k=-\infty}^{0} \lambda^{-k} \mathbf{D}_b \mathbf{P}_\pi^{-k} \mathbf{r}_\pi \\ &= \sum_{k=0}^{\infty} \lambda^{k} \mathbf{D}_b \mathbf{P}_\pi^{k} \mathbf{r}_\pi.\end{aligned}$$

3. This derivation is very similar to the previous one and uses the definition of $\mathbf{P}_\pi^\lambda$ (4.2).

□

Lemma 4.7. *Under Assumptions 4.1 and 4.5, the steady-state expectations* $\mathbf{A} = \mathbb{E}_d[\mathbf{A}_t]$ *and* $\mathbf{b} = \mathbb{E}_d[\mathbf{b}_t]$ *are:*

$$\begin{aligned}\mathbf{A} &= \mathbf{D}_b(\mathbf{P}_\pi^\lambda - \mathbb{I} - \frac{\eta}{1-\lambda}\mathbf{1}\mathbf{g}^\top), \\ \mathbf{b} &= \sum_{k=0}^{\infty} \lambda^k \mathbf{D}_b \mathbf{P}_\pi^k \mathbf{r}_\pi.\end{aligned}$$

It follows that $\mathbb{E}_{Y\sim d}[h(\mathbf{v}, Y)] = \mathbf{A}\mathbf{v} + \mathbf{b}$.

Proof. These follow directly from Lemma 4.6 and the expressions in (4.33) and (4.34).

□

Lemma 4.8. *If* η *is sufficiently small such that the off-diagonal elements of* $\mathbf{A}$ *are positive, then* $\mathbf{A}$ *is a Hurwitz matrix.*

Proof. In this proof, we first use a property of irreducible non-negative matrices and then that of the differential Bellman equations.

For any $\lambda > 0$, all elements of $\mathbf{P}_\pi^\lambda$ are positive under Assumption 4.1. If η is sufficiently small, then the off-diagonal elements of $\mathbf{A}$ are positive while the diagonal elements are negative. Adding $l\mathbf{I}$ to $\mathbf{A}$ with a large positive l ensures that the diagonal elements are also positive, making $\mathbf{A}+l\mathbf{I}$ a positive matrix. Since $\mathbf{A}+l\mathbf{I}$ is irreducible

and positive, by the Perron-Frobenius theorem (Horn & Johnson, 1985: Theorem 8.4.4), its spectral radius—the maximum magnitude of its eigenvalues—κ is a positive eigenvalue corresponding to a positive eigenvector $\mathbf{u}$. Now if $\{\lambda_j^{\mathbf{A}}\}$ denotes the set of all (possibly complex) eigenvalues of $\mathbf{A}$, then the eigenvalues of $\mathbf{A} + l\mathbf{I}$ are $\{\lambda_j^{\mathbf{A}} + l\}$. From the Perron-Frobenious theorem (and in particular, Horn & Johnson's (1985) Problem 9 at the end of Section 8.3):

$$\begin{aligned}
\kappa &\geq |\lambda_j^{\mathbf{A}} + l|, \quad \forall j \\
&= |(\mathrm{Re}(\lambda_j^{\mathbf{A}}) + l) + i\,\mathrm{Img}(\lambda_j^{\mathbf{A}})| \\
&\geq |\mathrm{Re}(\lambda_j^{\mathbf{A}}) + l| \\
&\geq \mathrm{Re}(\lambda_j^{\mathbf{A}}) + l \\
\implies \kappa - l &\geq \mathrm{Re}(\lambda_j^{\mathbf{A}}), \quad \forall j. \qquad (4.35)
\end{aligned}$$

Now we show that $\kappa - l$ is negative. We know that:

$$\begin{aligned}
(\mathbf{A} + l\mathbf{I})\mathbf{u} &= \kappa\mathbf{u}, \\
\mathbf{A}\mathbf{u} &= (\kappa - l)\mathbf{u}, \\
\mathbf{D}_b(\mathbf{P}_\pi^\lambda - \mathbb{I} - \frac{\eta}{1-\lambda}\mathbf{1}\mathbf{g}^\top)\mathbf{u} &= (\kappa - l)\mathbf{u}, \\
(\mathbf{P}_\pi^\lambda - \mathbb{I} - \frac{\eta}{1-\lambda}\mathbf{1}\mathbf{g}^\top)\mathbf{u} &= (\kappa - l)\mathbf{D}_b^{-1}\mathbf{u}, \\
\implies -(\kappa - l)\mathbf{D}_b^{-1}\mathbf{u} - \frac{\eta}{1-\lambda}\mathbf{1}\mathbf{g}^\top\mathbf{u} + \mathbf{P}_\pi^\lambda\mathbf{u} - \mathbf{u} &= \mathbf{0}. \qquad (4.36)
\end{aligned}$$

Note that this is the Bellman equation (2.7). The general form of the Bellman equation is $\mathbf{v} = \mathbf{r}_\pi - \bar{r}\mathbf{1} + \mathbf{P}_\pi\mathbf{v}$ and the solutions of $(\mathbf{v}, \bar{r})$ are of the form $(\mathbf{v}_\pi + c\mathbf{1}, r(\pi))$. Since $\mathbf{P}_\pi^\lambda$ is a transition matrix, the property of the average reward $r(\pi) = \mathbf{d}_\pi^\top\mathbf{r}_\pi$ implies:

$$\mathbf{d}_\pi^\top[-(\kappa - l)\mathbf{D}_b^{-1}\mathbf{u}] = \frac{\eta}{(1-\lambda)}\mathbf{g}^\top\mathbf{u}.$$

We know that $\mathbf{u}$ is a positive vector; we assumed $\eta > 0$ and that $\mathbf{g}$ is a non-negative vector with at least one positive element; Assumptions 4.1 and 4.5 imply $\mathbf{d}_\pi$ and $\mathbf{d}_b$

are positive vectors. This implies $\kappa - l < 0$. Substituting this in (4.35),

$$0 > \kappa - l \geq \mathrm{Re}(\lambda_j^{\mathbf{A}}), \quad \forall j.$$

Thus, we have shown the real parts of all the eigenvalues of $\mathbf{A}$ are negative. In other words, $\mathbf{A}$ is a Hurwitz matrix. □

Since $\mathbf{A}$ is Hurwitz, Khalil's (2002) Theorem 4.5 implies the ODE $\dot{\mathbf{u}} = \mathbf{A}\mathbf{u} + \mathbf{b}$ converges to the unique stable solution of $\mathbf{A}\mathbf{u} + \mathbf{b} = 0$ from any initial condition. It also implies the ODE $\dot{\mathbf{u}} = \mathbf{A}\mathbf{u}$ has the origin as the globally asymptotically stable equilibrium point.

The iterates are bounded and the algorithm converges

We now use general results from stochastic approximation theory by Borkar to show that sequence of discrete iterates $\{\mathbf{v}_t\}$ generated by (4.30) is stable and asymptotically tracks the solutions of the ODE. Theorem 4.4 is a specific instance of very general results by Borkar (2009).

Theorem 4.4 (Based on Borkar's (2009: Chapter 6) Theorem 9 and Corollary 8)**.** *Consider an iterative algorithm of the form:* $\mathbf{v}_{t+1} \doteq \mathbf{v}_t + \alpha_t\big[h(\mathbf{v}_t, Y_t) + \mathbf{m}_{t+1}\big]$. *Suppose the following conditions are satisfied:*

1. *The process* $\{Y_t\}$ *is a weak Feller Markov chain in a compact state space* $\mathcal{Y}$ *and has a unique invariant probability measure* d.
2. *The function* $h(\mathbf{v}, y)$ *is jointly continuous in* $(\mathbf{v}, y)$ *and is Lipschitz in* $\mathbf{v}$ *uniformly w.r.t.* $y \in \mathcal{Y}$.
3. *Define* $\tilde{h}(\mathbf{v}) \doteq \mathbb{E}_d\big[h(\mathbf{v}, Y)\big]$. *The limit* $\hat{h}(\mathbf{v}) \doteq \lim_{c\to\infty} \tilde{h}(c\,\mathbf{v})/c$ *exists uniformly on compact subsets of* $\mathbf{v}$. *The ODE* $\dot{\mathbf{u}} = \hat{h}(\mathbf{u})$ *is well posed and has the origin as the unique globally asymptotically stable solution.*
4. *The sequence* $\{\mathbf{m}_{t+1}\}$ *is a martingale difference sequence w.r.t. the increasing* σ*-fields* $\mathcal{F}_t \doteq \sigma(\mathbf{v}_k, Y_k, \mathbf{m}_k, k \leq t), t \geq 0$ *(that is,* $\mathbb{E}[\|\mathbf{m}_{t+1}\| \mid \mathcal{F}_t] < \infty$ *and*

$\mathbb{E}[\mathbf{m}_{t+1} \mid \mathcal{F}_t] = 0$ *almost surely,* $\forall t \geq 0$*), and* $\mathbb{E}[\|\mathbf{m}_{t+1}\|^2 \mid \mathcal{F}_t] < K(1 + \|\mathbf{v}_t\|^2)$ *almost surely,* $\forall t \geq 0$*, for some constant* $K > 0$*.*

5. *The step sizes* $\{\alpha_t\}$ *are positive with* $\sum_t \alpha_t = \infty$ *and* $\sum_t \alpha_t^2 < \infty$*.*

Then,

(i) the algorithm is stable, that is, $\sup_t \|\mathbf{v}_t\| < \infty$*, almost surely,*

(ii) the algorithm converges almost surely to a compact internally chain transitive invariant set of the ODE $\dot{\mathbf{u}} = \tilde{h}(\mathbf{u})$*.*

We now verify that the family of Differential TD(λ) algorithms satisfies the conditions of Theorem 4.4.

As mentioned earlier, it follows from Yu's (2012, 2017) analysis that the joint process $\{Y_t\}$ underlying Differential TD(λ) (4.30) is a weak Feller process with a unique invariant distribution. With the trace evolving in a bounded space due to Assumption 4.6, the overall process $\{Y_t\}$ evolves in a closed and bounded—*compact*—space (note that the states and actions sets $\mathcal{S}$ and $\mathcal{A}$ are finite discrete sets). Taken together, $\{Y_t\}$ evolves in a compact space $\mathcal{Y}$ with a unique invariant distribution d. This satisfies Condition 1.

It is easy to see that the function $h(\mathbf{v}, y)$ in (4.32) is jointly continuous in its arguments $\mathbf{v}$ and y. h is also Lipschitz in $\mathbf{v}$ uniformly w.r.t. $y \in \mathcal{Y}$ because we assumed the trace is bounded (Assumption 4.6). Hence, Condition 2 is satisfied.

For Condition 3, $\tilde{h}(\mathbf{v}) = \mathbf{A}\mathbf{v} + \mathbf{b}$ from Lemma 4.7. Then, $\hat{h}(\mathbf{v}) = \lim_{c\to\infty} \frac{\mathbf{A}(c\,\mathbf{v})+\mathbf{b}}{c} = \mathbf{A}\mathbf{v}$. From Lemma 4.8, there exists values of $\eta > 0$ such that the ODE $\dot{\mathbf{u}} = \mathbf{A}\mathbf{u}$ has the origin as the globally asymptotically stable equilibrium point. Hence, Condition 3 is satisfied.

With the first three conditions, Borkar's (2009) Theorem 8 guarantees that the iterates $\mathbf{v}_t$ are bounded, that is, $\sup_t \|\mathbf{v}_t\| < \infty$, almost surely.

For Condition 4, note that $\mathcal{F}_t$ contains all the history of Y_k and $\mathbf{m}_k$ for $k \leq t$.

Then,

$$\begin{aligned}\mathbb{E}[\mathbf{m}_{t+1} \mid \mathcal{F}_t] &= \mathbb{E}\big[\big(R_{t+1} - r(S_t, A_t, S_{t+1})\big)\mathbf{z}_t \mid \mathcal{F}_t\big] \\ &= \mathbb{E}[R_{t+1} \mid \mathcal{F}_t]\,\mathbf{z}_t - r(S_t, A_t, S_{t+1})\,\mathbf{z}_t \\ &= 0, \quad \forall t \geq 0.\end{aligned}$$

Next, because the rewards are bounded, the expected norm and the expected squared norm is bounded. Furthermore, we just showed the iterates $\{\mathbf{v}_t\}$ are bounded, so there exists some $K > 0$ such that $\mathbb{E}[\|\mathbf{m}_{t+1}\|^2 \mid \mathcal{F}_t] < K(1 + \|\mathbf{v}_t\|^2)$ almost surely, $\forall t \geq 0$. Hence, Condition 4 is satisfied.

Condition 5 is satisfied by Assumption 4.3.

Hence, the family of Differential TD(λ) algorithms (of which Algorithm 2 is an instance) satisfies all the conditions of Theorem 4.4. Because the matrix $\mathbf{A}$ in $\tilde{h}(\mathbf{v}) = \mathbf{A}\mathbf{v} + \mathbf{b}$ is a Hurwitz matrix, the compact internally chain transitive invariant set (see Borkar's (2009) Chapter 2 for precise definitions) of the ODE $\tilde{h}(\mathbf{v}) = \mathbf{A}\mathbf{v} + \mathbf{b}$ is just a single point satisfying $\mathbf{A}\mathbf{v} + \mathbf{b} = 0$. As a result, we conclude that the family of algorithms converges almost surely to that unique point $\mathbf{v}_\infty$.

The fixed point is a solution of the Bellman equations

We now check that the fixed point is a solution of the differential multi-step Bellman equations (4.3):

$$\begin{aligned}0 &= \mathbf{A}\mathbf{v}_\infty + \mathbf{b} \\ &= \Big(\mathbf{D}_b(\mathbf{P}_\pi^\lambda - \mathbb{I} - \frac{\eta}{1-\lambda}\mathbf{1}\mathbf{g}^\top)\Big)\mathbf{v}_\infty + \Big(\sum_{k=0}^{\infty}\lambda^k\mathbf{D}_b\mathbf{P}_\pi^k\mathbf{r}_\pi\Big) \\ &= (\mathbf{P}_\pi^\lambda - \mathbb{I} - \frac{\eta}{1-\lambda}\mathbf{1}\mathbf{g}^\top)\mathbf{v}_\infty + \sum_{k=0}^{\infty}\lambda^k\mathbf{P}_\pi^k\mathbf{r}_\pi \\ \implies \mathbf{v}_\infty &= \sum_{k=0}^{\infty}\lambda^k\mathbf{P}_\pi^k\mathbf{r}_\pi - \frac{\eta}{1-\lambda}\mathbf{1}\mathbf{g}^\top\mathbf{v}_\infty + \mathbf{P}_\pi^\lambda\mathbf{v}_\infty.\end{aligned}$$

The last equation is the multi-step differential Bellman equation (4.3). Hence, $\mathbf{v}_\infty$ is a solution of the differential Bellman equation with $\eta\,\mathbf{g}^\top\mathbf{v}_\infty = r(\pi) = f(\mathbf{v}_\infty)$.

This concludes the proof of Theorem 4.3. □

4.6 Off-policy: Experiments

In this section, we test the efficacy of Algorithm 2 (4.23–4.26), which is an instance of the family of off-policy Differential TD(λ) algorithms. In particular, we would like to validate that the algorithm converges. From the theory, we also expect the benefit of multi-step methods to decrease as the extent of the problems' *off-policy-ness* increases. I use the term 'off-policy-ness' to mean the difference between the behavior and target policies. As the behavior policy becomes increasingly different from the target policy, the magnitude of the largest importance-sampling (IS) ratio increases, which can increase the variance of the multi-step updates.

We created several off-policy problems using the 19-state random-walk domain from Section 4.3. The target policy took both the left and right actions with equal probability. Five different behavior policies took the right action with probabilities $b[\text{right}|\cdot] \in \{0.5, 0.45, 0.4, 0.35, 0.3\}$ to create a set of five problems with increasing off-policy-ness. Table 4.1 shows the value of the largest IS ratio for each of the five problems and the corresponding largest value of λ for which Theorem 4.3 guarantees convergence (recall Assumption 4.6).

We applied Algorithm 2 to all five problems with the same experimental setting as in Section 4.3. That is, we tested $\alpha \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,$

Table 4.1: The largest importance-sampling ratio corresponding to each off-policy problem and the corresponding largest value of λ for which Theorem 4.3 guarantees convergence.

$b[\text{right}\|\cdot]$	$\rho_{\max}$	$\lambda_{\max}$
0.5	1	1
0.45	10/9	0.9
0.4	10/8	0.8
0.35	10/7	0.7
0.3	10/6	0.6

0.9, 1.0}, $\eta \in$ {0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0}, and $\lambda \in$ {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99}. Each parameter configuration was run for 100 independent runs of 10,000 time steps. The step sizes were decayed by a factor of 0.99975 at each time step. The weight vector, trace vector, average-reward trace parameter, and the average-reward estimate were all initialized to zero. We also used the same evaluation metric: root-mean-squared value error w.r.t. the nearest solution of the Bellman equations, referred to as 'RMSVE (TVR)'.

Figure 4.5 shows the sensitivity plots of Algorithm 2 in comparison to Algorithm 1. The sensitivity plot of Algorithm 1 is replicated from Figure 4.3 for ease of comparison. Each point denotes the value error over the first 2000 steps averaged across the 100 independent runs; the error bars denote one standard error of the mean.

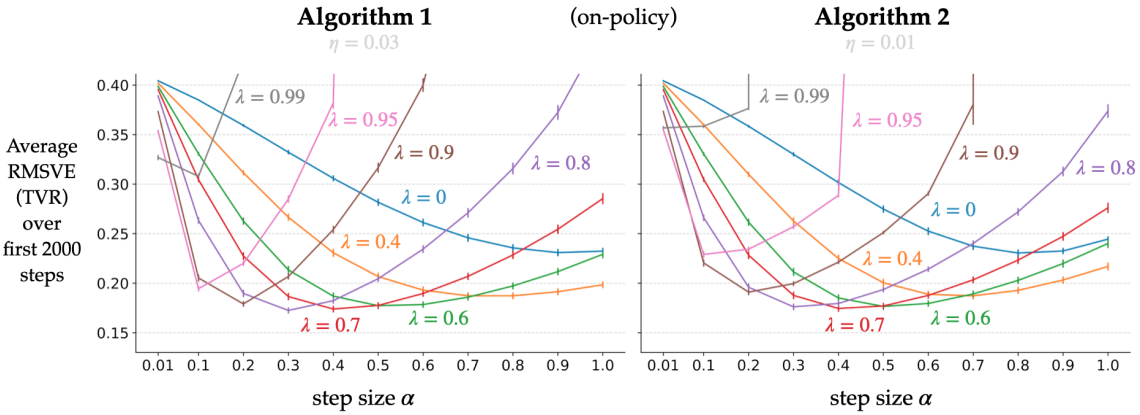


Figure 4.5: Sensitivity of the performance of Algorithm 1 and Algorithm 2 w.r.t. their parameters α and λ on the on-policy 19-state random-walk problem.

The performance of both algorithms was quite similar on this problem, which indicates that Algorithm 2—designed for the off-policy setting—works well even in the on-policy setting. This is not surprising because in the on-policy setting the scalar reward-rate trace saturates quickly to $1/(1-\lambda)$, in effect, scaling the step-size parameter η in the reward-rate update. Such a scaling would not affect the performance much, just like we observed. The plot for Algorithm 2 corresponds to a smaller η than that of Algorithm 1 to account for the scaling factor; the trends were not too different for other values of η.

Figure 4.6 shows learning curves corresponding to different values of λ across three problems. The leftmost plot is for the on-policy problem; the other two plots correspond to increasing disparity between the behavior and the target policies. We plotted the learning curves for five values of λ for each problem: $\lambda = 0$ as a baseline, the largest value of λ that did not result in divergence, and three values in between. The curves correspond to the values of α and η that resulted in the lowest value error across the training period.

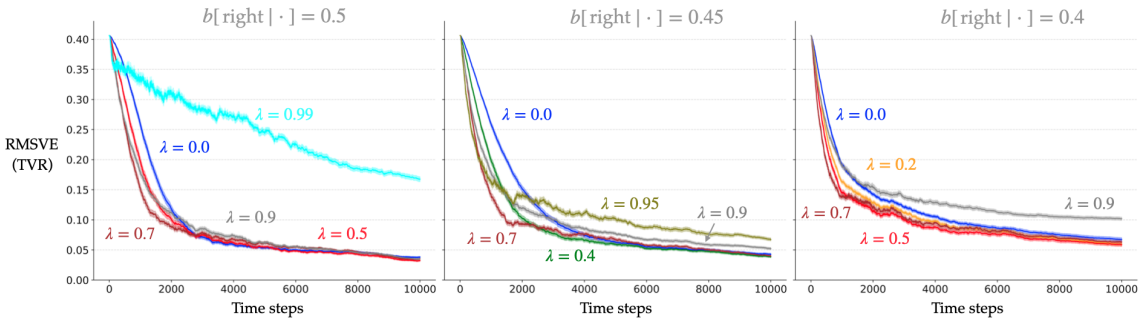


Figure 4.6: Learning curves for Algorithm 2 on different problems corresponding to five values of λ.

The algorithm approached convergence for several values of λ on each problem (approaching zero error took about 30k steps; only 10k steps are shown here to highlight the difference in early performance). The average-reward estimate also converged to its true value of zero; as before, its learning curves from an initialization of zero are relatively uninformative and hence not shown here. Again, an intermediate value of λ resulted in a good balance of learning speed and variance in updates.

In addition, the largest value of λ that did not result in divergence decreased from 0.99 to 0.95 to 0.9 as the behavior policy became increasingly different from the target policy. Note that these values of λ are larger than the ones for which Theorem 4.3 guarantees convergence (see Table 4.1). This is because the convergence result accounts for the worst-case scenario which may not always occur in practice.

Another thing to note is the rate of convergence for a particular value of λ as the problems' off-policy-ness increases. For example, from left to right in Figure 4.6, for $\lambda = 0.9$, it takes longer to reach an error rate of 0.1. The difference in the rate of learning compared to the $\lambda = 0$ case also decreased from left to right. This difference indicates that the increased variance due to the eligibility trace in off-policy problems counters the trace's potential benefits of multi-step credit assignment. Figure 4.7 further highlights this trend for $\lambda = 0.6$ (corresponding to the values of α and η that resulted in the lowest value error across the training period). The value error decreased quickly at first along the states that occur frequently under the behavior policy (the left side of the 19-state chain) but the rate of reduction became lower as the off-policy-ness increased.

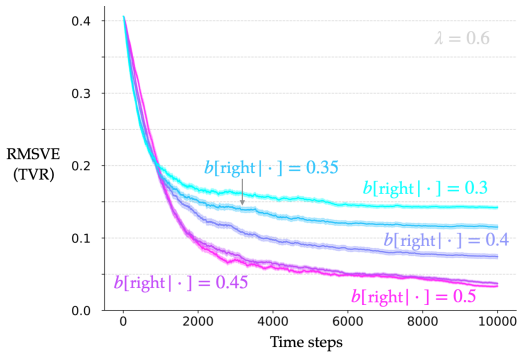


Figure 4.7: Learning curves for Algorithm 2 corresponding to $\lambda = 0.6$ on the five different off-policy problems.

Finally, Figure 4.8 shows the sensitivity of the average performance of Algorithm 2 w.r.t. its parameters α and η for a fixed value of $\lambda = 0.6$. From left to right, we again saw that for a given value of η, the rate of learning was slower as the off-policy-ness increased. Furthermore, the value of η with low value error across step sizes was

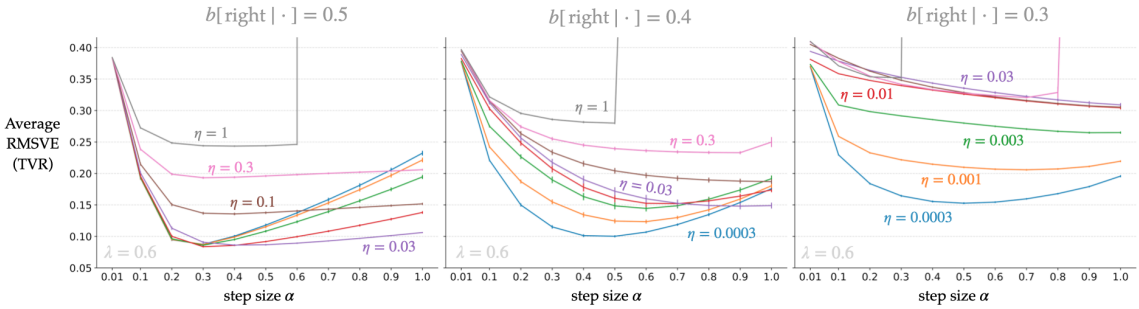


Figure 4.8: Sensitivity of the performance of Algorithm 2 w.r.t. its parameters α and η on three of the off-policy problems.

relatively larger for the on-policy problem than the off-policy problems.

From these simple experiments, we noted that Algorithm 2—an instance of the family of off-policy Differential TD(λ) methods—works as intended: the value and average-reward estimates to a solution of the differential Bellman equations (2.6). Like in the on-policy setting, an intermediate value of λ can lead to a good trade-off between the speed of learning and the variance in updates. However, such a good value of λ shifts closer towards zero as the problems' degree of off-policy-ness increases.

4.7 Discussion and Conclusion

In this chapter, I introduced two multi-step average-reward prediction algorithms that extend the one-step *differential* methods introduced in Chapter 3 and use the TD error to update their average-reward estimates. In addition, I showed:

1. Algorithm 1 converges in the *on-policy* setting with *linear* function approximation.

2. The family of algorithms of which Algorithm 2 is a member converges in the *tabular off-policy* setting.

The latter family of algorithms is the first set of multi-step average-reward algorithms that are proved to converge in the off-policy setting, and hence represent an important addition to the average-reward RL literature.

I validated the theoretical results through simple but pointed experiments. The experimental results also show that the intuitions about trace-decay parameter λ carry over from the total- or discounted-reward settings: an intermediate value of λ typically results in a good balance between increased speed of learning and increased variance of updates.

The work in this chapter constitutes key initial steps in completely fleshing out multi-step algorithms for the average-reward formulation. There are several directions

in which future research can proceed. The first and most obvious need is the extension of the off-policy proof (of Theorem 4.3) to include the $\lambda = 0$ case. To show that the key matrix $\mathbf{A}$ is Hurwitz, we used the property that all elements of the matrix $\mathbf{P}_\pi^\lambda$ are positive, which may not in general be true when $\lambda = 0$. Hence the theorem statement considers specifies $\lambda > 0$. An extension to the $\lambda = 0$ case should be straightforward, though it is not obvious. Such an extension may also enable the choice of η to be arbitrary rather than being "sufficiently small", as is required now for $\mathbf{A}$ to be Hurwitz.

Speaking of conditions of the proof, the convergence result is likely applicable to a broader family of algorithms than the ones we specified. Our proof (in its current form) applies to cases where the self-referential function f is of the form $f(\mathbf{v}_t) = \eta \mathbf{g}^\top \mathbf{v}_t$, with $\eta > 0$ and $\mathbf{g} \in \mathbb{R}^{|\mathcal{S}|}$ is a non-negative vector with at least one positive element (Assumption 4.4). However, we only require $f(\mathbf{u})$ to be positive, where $\mathbf{u}$ is a positive vector. So f can be a non-linear function of $\mathbf{u}$, such as max or min. It would be useful to flesh out all the complete family of algorithms for which the convergence proof applies in the off-policy setting.

The family of algorithms developed in this chapter is similar to the family of Abounadi, Bertsekas, and Borkar's (2001) RVI Q-learning algorithms. As with the latter family, it is not clear which member of the former family should a practitioner use for off-policy average-reward problems. Algorithm 2 is just one instance of the family. We highlighted it because it has a clear and elegant interpretation in terms of a trace parameter, which may not be possible for other members of the family. It is pertinent to investigate other members of the family, in terms of performance as well as computational-efficiency considerations.

The convergence proof for Algorithm 2 and its family is restricted to the tabular case. An important direction of future work involves developing algorithms and convergence proofs for the multi-step off-policy average-reward setting with (at least linear) function approximation. As discussed in Section 4.4, this is an active area

of research in the discounted-reward formulation. Zhang et al. (2021) extended the *one-step* version of Gradient TD (Maei et al., 2010) to the average-reward formulation. He et al. (2022) have worked on preliminary average-reward extensions of the Emphatic-TD approach (Sutton et al., 2016). Much work remains.

This chapter takes some of the first steps in developing the multi-step average-reward literature to the extent of the discounted-reward literature. It would be interesting to explore equivalents of True Online TD(λ) (van Seijen et al., 2016) and Tree-Backup(λ) (Precup et al., 2000), or extensions to the variable-λ case (e.g., Yu, 2012). Extensions of the proposed algorithms to the control case are simple (e.g., we present the pseudocode for Differential Sarsa(λ) in Appendix A), however, convergence proofs are expected to be more challenging.

Chapter 5

Reward Centering

So far we have discussed the average-reward formulation for continuing problems. The *discounted-reward* formulation offers another way to make a sum of infinite rewards finite in continuing problems.[1] The formulation is also applicable in episodic problems, where it has been used extensively, including within several impressive demonstrations of RL (e.g., Mnih et al., 2015; Silver et al., 2018; Bellemare et al., 2020; Wurman et al., 2022; Kaufmann et al., 2023).

In this chapter, we show that the simple idea of estimating and subtracting the average reward from the observed rewards can lead to a significant improvement in performance when using standard discounting methods such as TD-learning or Q-learning on continuing problems. The improvement becomes larger as the discount factor approaches one. The underlying theory dates back to 1962 with Blackwell's seminal work on dynamic programming in discrete MDPs. However, we are still realizing some of its deeper implications. I discuss two in particular:

1. Mean-centering the rewards removes a state-independent constant (that scales inversely with $1 - \gamma$) from the value estimates, enabling the value-function approximator to focus on the differences between the states (or actions within states). As a result, values corresponding to discount factors arbitrarily close

[1] We specifically mean geometric discounting in which a reward n steps from now is weighted γ^n lower than the current reward, where $\gamma \in [0, 1)$. The sum of rewards may not be finite with other forms of discounting such as hyperbolic discounting (Fedus et al., 2019), which are applicable only in terminating episodic problems.

to 1 can be estimated relatively easily (e.g., without any numerical instability).

2. Furthermore, mean-centering the rewards (unsurprisingly) makes standard methods robust to any constant offset in the rewards. This can be useful when applying RL algorithms in problems where the properties of the reward signal are unknown or changing.

In the simpler prediction problem, we first consider the most obvious mechanism of reward centering: maintain a running average of the observed rewards. This simple idea is highly effective. However, it is limited to the on-policy setting. For the off-policy setting, we take inspiration from Chapter 3 where we showed that the average reward can be estimated in the off-policy setting using the TD error. We then assess the effects of reward centering in the control problem via an empirical case study using the tabular, linear, and non-linear variants of the Q-learning algorithm. Towards the end, I discuss the connections between reward centering and related approaches that modify the rewards or consider some kind of decomposition.

5.1 The Idea

Reward centering is a simple idea: subtract the average reward from the observed rewards. Doing so makes the rewards appear *mean-centered*. The average reward can be estimated empirically.

This idea is not new. The effect of centered rewards is well known in the bandit setting. For instance, Sutton and Barto (2018: Section 2.8) demonstrated that estimating and subtracting the average reward from the observed rewards can significantly affect the rate of learning.

The core idea behind centering is revealed by the Laurent-series decomposition of the *discounted* value function. In the tabular case, the discounted value function $v_\pi^\gamma : \mathcal{S} \to \mathbb{R}$ for a policy π corresponding to a discount factor $\gamma \in [0, 1)$ can be

decomposed as:

$$v_\pi^\gamma(s) = \frac{r(\pi)}{1-\gamma} + \tilde{v}_\pi(s) + e_\pi^\gamma(s), \qquad \forall s, \tag{5.1}$$

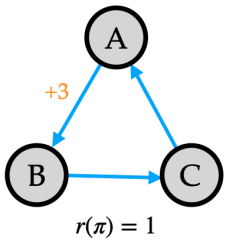
where, $r(\pi)$ is the average reward obtained by policy π (that induces a unichain; see Chapter 2), $\tilde{v}_\pi : \mathcal{S} \to \mathbb{R}$ denotes the *differential* value function,[2] and $e_\pi^\gamma(s)$ denotes an error term that goes to zero as the discount factor goes to one (Blackwell, 1962: Theorem 4a; also see Puterman's (1994) Corollary 8.2.4). This decomposition for state values (5.1) also implies a similar decomposition for action values:

$$q_\pi^\gamma(s,a) = \frac{r(\pi)}{1-\gamma} + \tilde{q}_\pi(s,a) + e_\pi^\gamma(s,a), \qquad \forall s, a. \tag{5.2}$$

The Laurent-series decomposition shows that the discounted value function includes a constant state(–action)-independent term. This term explains the improvements in the bandit setting (see Sutton & Barto's (2018) Figure 2.5). The action value estimates (of the single state) are initialized to zero and the true values are close to +4. Even though the relative values of the actions is all that matters for action selection, each action-value estimate has to learn the action-independent offset. Approximation errors in estimating the offset can easily mask the relative differences in actions. This is especially true when we consider the full RL problem with a non-zero discount factor because the offset scales inversely with $(1-\gamma)$.

To build some intuition, let us start with the prediction setting. Consider the three-state Markov reward process (MRP) depicted in Figure 5.1 induced by a policy π in some MDP. There is a reward of +3 on going from state s_A to s_B; 0 otherwise. The average reward $r(\pi)$ is 1. The discounted state values for three discount factors are shown in Figure 5.1. Note the magnitude of the values and especially the jump when the discount factor is increased. Now consider the values with the constant offset subtracted from each state, $v_\pi^\gamma(s) - r(\pi)/(1-\gamma)$, which we call the *centered* discounted values: $\tilde{v}_\pi^\gamma$. The magnitudes of the centered values are much smaller, and

[2]Note that we are now using $\tilde{v}_\pi$ to denote the differential value function instead of v_π as the previous chapters. The reason will be evident shortly.



	$\frac{r(\pi)}{1-\gamma}$
$\gamma = 0.8$	5
$\gamma = 0.9$	10
$\gamma = 0.99$	100

		s_A	s_B	s_C
Standard discounted values	$\gamma = 0.8$	6.15	3.93	4.92
	$\gamma = 0.9$	11.07	8.97	9.96
	$\gamma = 0.99$	101.01	98.99	99.99
Centered discounted values	$\gamma = 0.8$	1.15	-1.07	-0.08
	$\gamma = 0.9$	1.07	-1.03	-0.04
	$\gamma = 0.99$	1.01	-1.01	-0.01
Differential values		1	-1	0

Figure 5.1: Comparison of the standard discounted values and the centered discounted values on a simple three-state problem.

differ only slightly when the discount factor is increased. The differential values are also shown for reference. These trends hold in general: for a given problem, the magnitude of the discounted values shoots up as the discount factor approaches 1; meanwhile, the centered discounted values do not change much and approach the differential values.

Formally, the centered discounted value function is the expected discounted sum of mean-centered rewards:

$$\tilde{v}_\pi^\gamma(s) \doteq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \big(R_{t+1} - r(\pi)\big) \mid S_t = s, A_{t:\infty} \sim \pi\right], \tag{5.3}$$

where $\gamma \in [0, 1]$. When $\gamma = 1$, the centered discounted values are the same as the differential values, that is, $\tilde{v}_\pi^\gamma(s) = \tilde{v}_\pi(s), \forall s$. More generally, the centered discounted values are the differential values plus the error terms from the Laurent-series decomposition (5.1):

$$v_\pi^\gamma(s) = \frac{r(\pi)}{1-\gamma} + \underbrace{\tilde{v}_\pi(s) + e_\pi^\gamma(s)}_{\tilde{v}_\pi^\gamma(s)}, \qquad \forall s,$$

Reward centering thus enables partitioning the information contained in the discounted value function into two parts: the state-independent average reward and the centered discounted value function. Such a partition can be useful:

a. As $\gamma \to 1$, the discounted values tend to explode but the centered discounted values remain small and tractable.

b. If the problems' rewards are shifted by a constant c, the magnitude of the discounted values increases by $c/(1-\gamma)$. The centered discounted values are unchanged; the average reward increases by c.

Another intriguing implication is if we want to increase discount factor (which is an algorithm parameter here) within the lifetime of a learning agent. The discounted values can change massively (see, e.g., Figure 5.1); the centered discounted values would not change much, and the changes become minuscule as the discount factor approaches 1.

The prerequisite to these potential benefits is the estimation of the average reward from data. Even the simplest idea can take us quite far.

5.2 Simple Reward Centering

The simplest way to estimate the average reward is to maintain a running average of the rewards observed so far. That is, if $\bar{R} \in \mathbb{R}$ denotes the estimate of the average reward, after t time steps, $\bar{R}_t = \sum_{k=1}^{t} R_k$. More generally, the estimate can be updated with a step-size parameter β_t:

$$\bar{R}_{t+1} \doteq \bar{R}_t + \beta_t(R_{t+1} - \bar{R}_t). \tag{5.4}$$

As discussed in Chapter 3, this is an unbiased estimate of the average reward corresponding to the policy π with which the data is generated (that is, in the on-policy setting).

To estimate the centered discounted values for a given policy π, we can use standard algorithms such as TD-learning (Sutton, 1988a), with the rewards centered using the current estimate of the average reward. For example, after the transition $(S_t, A_t, R_{t+1}, S_{t+1})$, the centered value estimates $\tilde{V}^\gamma : \mathcal{S} \to \mathbb{R}$ can be updated as:

$$\tilde{V}_{t+1}(S_t) \doteq V_t(S_t) + \alpha_t\big[(R_{t+1} - \bar{R}_t) + \gamma \tilde{V}_t^\gamma(S_{t+1}) - \tilde{V}_t^\gamma(S_t)\big]. \tag{5.5}$$

Let us call the algorithm that updates the value and average-reward estimates as in (5.4) and (5.5) as *TD-learning with reward centering*. I conjecture that a convergence proof for this on-policy setting should be straightforward: under a two-timescale argument, convergence of the average-reward estimate would imply the convergence of the value estimates under the same conditions as Sutton's TD-learning.

Tabular on-policy TD(0) learning with reward centering

At time step t, with the knowledge of $(S_t, A_t, R_{t+1}, S_{t+1})$, update the value and average-reward estimates as:

$$\tilde{V}^{\gamma}_{t+1}(S_t) \doteq \tilde{V}^{\gamma}_{t}(S_t) + \alpha_t\big(R_{t+1} - \bar{R}_t + \gamma\tilde{V}^{\gamma}_{t}(S_{t+1}) - \tilde{V}^{\gamma}_{t}(S_t)\big),$$
$$\bar{R}_{t+1} \doteq \bar{R}_t + \eta\alpha_t(R_{t+1} - \bar{R}_t),$$

where, like in Chapter 3, $\eta > 0$ is a step-size parameter for the average-reward estimate.

We tested the efficacy of this simple approach to reward centering. Primarily, we wanted to check if learning the average reward and the value estimates separately indeed results in a higher rate of learning. Secondarily, we wanted to test if the benefits increase as the discount factor approaches 1.

We began our investigation on a variant of the RandomWalk domain used in Chapter 4. There are seven states; the right action from the rightmost state leads to the middle state with reward +7 and the left action from the leftmost state leads to the middle state with reward +1. The target policy is the one that takes both actions in each state with equal probability, that is, $\pi(\text{left}|\cdot) = \pi(\text{right}|\cdot) = 0.5$. The average reward corresponding to this policy is 0.25.

In the first on-policy experiment, we tested three variants of the TD-learning algorithm: (1) standard TD-learning, (2) TD-learning with rewards that are centered by an oracle (i.e., the average reward is somehow known—not estimated), (3) TD-

learning with reward centering using (5.4). We performed the same experiment with two discount factors, $\gamma = 0.9$ and 0.99. Each algorithm was run for 50,000 steps and repeated 50 times each. The step size α was decayed by 0.99999 at each step. The values estimates for all variants and the average-reward estimate for TD-with-centering were initialized to zero.

We evaluated the root mean-squared value error of the estimates and the true discounted values w.r.t. the steady-state distribution of the states induced by the target policy π. The centering and oracle-centered methods estimate the centered discounted value function $\tilde{\mathbf{v}}_\pi^\gamma$, so for a one-to-one comparison, we added $\bar{R}/(1-\gamma)$ to the centered estimates to compute the uncentered values.

Figure 5.2 shows the learning curves for this on-policy experiment, one plot each for the two different values of γ. We tested $\alpha \in \{0.01, 0.02, 0.04, 0.08, 0.16, 0.32\}$ and picked the one which resulted in the lowest average RMSVE across the training period for standard uncentered approach ($\alpha = 0.04$ for $\gamma = 0.9$ and $\alpha = 0.08$ for $\gamma = 0.99$). Corresponding to these step sizes, we plotted the learning curves for the simple centering approach for two values of η out of the four tested $(1/640, \mathbf{1/160}, \mathbf{1/40}, 1/10)$. Each solid point represents the RMSVE averaged over the 50 independent runs; the

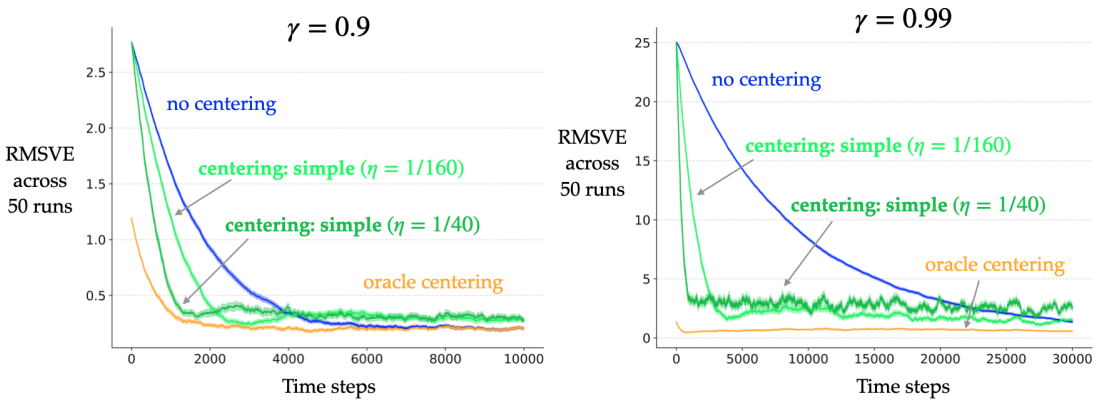


Figure 5.2: Learning curves demonstrating the performance of standard TD-learning (blue), TD-learning with rewards that are centered by an oracle (orange), and TD-learning with centering (green) on a seven-state variant of the continuing RandomWalk problem. Note the difference in the scales of two axes.

shaded region denotes one standard error.

Let us first consider the plot on the left corresponding to $\gamma = 0.9$. For the uncentered and the centering approach, the learning curve starts at just over 2.5. One can verify this by computing the true discounted values for $\gamma = 0.9$. Alternatively, we can get a quick estimate using the Laurent-series decomposition (5.1) which says that all the values have a state-independent constant of $r(\pi)/(1-\gamma)$—in this case, $0.25/(1-0.9) = 2.5$. The oracle-centered learning curve starts much lower because it magically has a fixed average-reward estimate of 0.25 from the start. The first thing to note is that standard TD-learning eventually converges to the same error rate as the oracle-centered version, which is expected. Learning the average reward and subtracting it indeed helps reduce the RMSVE much faster in the beginning compared to when there is no centering. However, the eventual error rate is higher. This is also expected because the average-reward estimate is changing over time, leading to more variance in the updates compared to the uncentered or oracle-centered version. Similar trends hold for the larger discount factor (right of Figure 5.2), where the uncentered approach now appears much slower in comparison.

One important observation from the plot on the right is when η is larger, the RMSVE reduction is higher in the beginning; however, this also results in a larger asymptotic error rate. This suggests the use of a step-size adaptation technique for the average-reward estimate, which sets a larger step size when the errors are large and prevalent and scales them down otherwise. We do not explore step-size adaptation in this chapter; this is an appealing direction for future work. However, we verified that the reward-centering approaches indeed learn an average-reward estimate that is around 0.25 on average in both experiments.

These simple experiments verify that the simple reward-centering technique can be quite effective in the on-policy setting, and the effect is more pronounced for larger discount factors.

The off-policy setting

What about the off-policy setting? We know that (5.4) leads to an unbiased estimate of the behavior policy's average reward. In the off-policy setting, the behavior policy b is different from the target policy π, which implies that $\bar{R}$ will converge to $r(b)$, not to $r(\pi)$. We also know that using the appropriate importance-sampling ratio in the update is not enough to guarantee convergence to $r(\pi)$ (see Section 3.1).

Let us analyze the impact of an inaccurate average-reward estimate. First, note that the centered discounted value function also satisfies a recursive Bellman-like equation:

$$\tilde{v}^\gamma(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r - \bar{r} + \gamma \tilde{v}^\gamma(s')\big],$$

$$\text{or, in vector notation,} \quad \tilde{\mathbf{v}}^\gamma = \mathbf{r}_\pi - \bar{r}\mathbf{1} + \gamma \mathbf{P}_\pi \tilde{\mathbf{v}}^\gamma, \tag{5.6}$$

where, $\tilde{\mathbf{v}}^\gamma$ denotes a vector in $\mathbb{R}^{|\mathcal{S}|}$, $\mathbf{r}_\pi$ is a vector of the expected one-step reward from each state, $\bar{r}$ is a scalar variable, $\mathbf{1}$ is a vector of all ones, and $\mathbf{P}_\pi$ is the state-to-state transition matrix induced by the policy π. It is easy to verify that the solutions of (5.6) are of the form $\big(\tilde{\mathbf{v}}^\gamma_\pi + c\mathbf{1}, r(\pi) - c(1-\gamma)\big), \forall c \in \mathbb{R}$, where $\tilde{\mathbf{v}}^\gamma_\pi$ denotes the centered differential value function (5.3) corresponding to policy π and discount factor γ.

We can equivalently write the family of solutions as $\big(\tilde{\mathbf{v}}^\gamma_\pi + \frac{k}{1-\gamma}\mathbf{1}, r(\pi) - k\big), \forall k \in \mathbb{R}$. This shows that if the average-reward estimate is off by k, then the centered discounted values each have a constant offset of $k/(1-\gamma)$. This is undesirable. The primary motivation of reward centering is to eliminate the state(–action)-independent offset from the estimates.[3] So we are motivated to find a way to estimate the target policy's average reward while behaving according to a different behavior policy.

However, note that an inaccurate estimation of the average reward is not a deal-breaker: standard algorithms that do not center the rewards can be perceived as using

[3]Consider the case when the average-reward estimate is zero, that is $k = r(\pi)$. The corresponding solution of the value estimates is then $\tilde{\mathbf{v}}^\gamma_\pi + \frac{r(\pi)}{1-\gamma}$, which is the standard (uncentered) discounted value function $\mathbf{v}^\gamma_\pi$.

a fixed inaccurate estimate of the average reward (zero), yet they are guaranteed (at least in the tabular case) to converge to the values of the target policy. So the issue is less about convergence and more about the rate of learning. Estimating the average reward accurately may yield better sample-complexity bounds when using standard methods than simply estimating the uncentered values (e.g., the bounds for Q-learning involve powers of $1/(1-\gamma)$ (Qu & Wierman, 2020; Wainwright, 2019; Even-Dar et al., 2003)).

We have seen in Figure 5.1 that when the rewards are centered by an oracle, the rate of learning is much higher compared to when there is no centering. If $\bar{R} = r(\pi)$ is the ideal value that results in the highest (relative) rate of learning, then $\bar{R} \in \big(0, 2r(\pi)\big)$ should roughly result in a higher rate of learning than the baseline rate of uncentered algorithm that sets $\bar{R} = 0$ (assuming $r(\pi) > 0$ without loss of generality). Of course, the average reward is estimated from data, which might nullify increases in the rate of learning at the edges of above boundaries.

In summary, the simple method of reward centering (5.4) can be highly effective when the average reward of the behavior policy is close to that of the target policy. Generally speaking, this may be true when the two policies are similar, like a greedy target policy and an ϵ-greedy behavior policy with a relatively small value of ϵ. However, the benefits of reward centering in terms of rate of learning may reduce and even disappear as the difference in the two policies increases. In the next section, we consider an alternative method to estimate the average reward more accurately in the off-policy setting.

5.3 Value-based Reward Centering

The problem of estimating the average reward in the off-policy setting is a familiar one by now—we encountered it in the previous chapters. We showed in Chapter 3 that we can obtain an unbiased estimate of the average reward in the off-policy setting by updating the average-reward estimate with the current differential TD

error—δ_t—instead of the error between the current reward and the current average-reward estimate—$(R_{t+1} - \bar{R}_t)$.

It turns out that this TD-error idea from the average-reward formulation is quite effective even in the discounted-reward formulation, which is the focus of the current chapter. In particular, we can obtain a relatively accurate estimate of $r(\pi)$ as long as the behavior policy b has coverage (recall Assumption 4.5 in Chapter 4). The estimation is not completely accurate; however, the estimation approaches the true value as the number of state–action pairs in the problem increases. Before we precisely characterize the quality of this approximation (in the next section), let us consider the approach in more detail.

The TD error for the discounted-reward formulation with centering is, at time step t, $\delta_t \doteq \big(R_{t+1} - \bar{R}_t + \gamma \tilde{V}^\gamma_t(S_{t+1}) - \tilde{V}^\gamma_t(S_t)\big)$. The idea is to update the average-reward estimate $\bar{R}_t$ using this TD error instead of the conventional error $R_{t+1} - \bar{R}_t$. Since this centering approach now involves values in addition to the reward, we call it *value-based centering*.

Tabular TD(0) with value-based reward centering

At time step t, with the knowledge of $(S_t, A_t, R_{t+1}, S_{t+1})$, update the value and average-reward estimates as:

$$\tilde{V}^\gamma_{t+1}(S_t) \doteq \tilde{V}^\gamma_t(S_t) + \alpha_t \rho_t \delta_t, \tag{5.7}$$

$$\bar{R}_{t+1} \doteq \bar{R}_t + \eta \alpha_t \rho_t \delta_t, \tag{5.8}$$

where, $\delta_t \doteq R_{t+1} - \bar{R}_t + \gamma \tilde{V}^\gamma_t(S_{t+1}) - \tilde{V}^\gamma_t(S_t)$, and $\rho_t \doteq \dfrac{\pi(A_t|S_t)}{b(A_t|S_t)}$.

We present the convergence result for the control variant of this algorithm in the next section: Q-learning with value-based reward centering. The convergence result for this prediction algorithm should be relatively straightforward.

We can gain some interesting insights about centering in the off-policy setting with simple experiments. Using the same continuing RandomWalk domain as in the previous section, we created the off-policy problem of evaluating the target policy that takes random actions in each state with equal probability: $[\pi(\text{left}|\cdot), \pi(\text{right}|\cdot)] = [0.5, 0.5]$. We used two behavior policies: $[b_1(\text{left}|\cdot), b_1(\text{right}|\cdot)] = [0.7, 0.3]$, $[b_2(\text{left}|\cdot), b_2(\text{right}|\cdot)] = [0.3, 0.7]$. The evaluation metric was the same as in the on-policy experiment—the root mean-squared value error (RMSVE) of the estimates and the target policy's values w.r.t. the state distribution induced by the target policy. We tested off-policy TD-learning (5.7) with both kinds of reward centering: the simple kind with an importance-sampling ratio in the update (5.4), and value-based centering (5.8). Each parameter configuration was run for 50,000 steps and repeated 50 times. As baselines, we also ran standard off-policy TD (a) without centering, and (b) with rewards centered by an oracle. In addition, we also tried value-based centering in the on-policy experiments from the previous section.

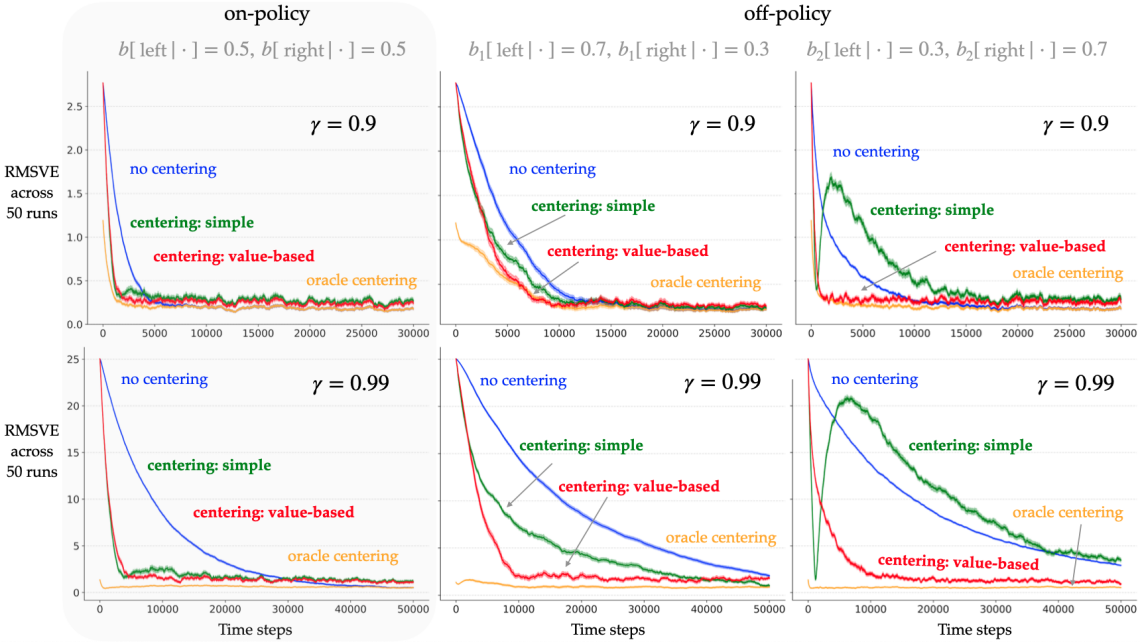


Figure 5.3: Learning curves demonstrating the performance of standard TD-learning (blue), TD-learning with rewards that are centered by an oracle (orange), TD-learning with simple centering (green), and TD-learning with value-based centering (red) on on- and off-policy problems for two discount factors.

Figure 5.3 shows learning curves for these experiments: each column shows the plots for a particular on- or off-policy problem; the two rows correspond to two discount factors.

First, consider the on-policy plots. Value-based centering appears as good as simple centering; more importantly, it does not hurt in the on-policy setting.

Next, consider the off-policy plots. The two different behavior policies are symmetric but result in different trends. Recall that the left side has the smaller reward. The rightmost state (with the larger reward) has the highest value, so it contributes more to the RMSVE compared to the leftmost state (note the two extreme states have the same weighting and the initial estimates are zero). b_1 results in the agent spending more time in the left side of the Markov chain, hence the reduction in RMSVE is relatively slower compared to when behaving with b_2.

Corresponding to b_1, we saw that value-based centering reached a lower RMSVE faster than simple centering, but the final error rate was the same for both centering approaches. This is expected; in the previous section we discussed that the centered discounted value function has infinite solutions corresponding to different values of the average-reward estimate. Something more interesting happened with b_2. The RMSVE reduced rapidly at first with simple centering, then rose, and reduced again. This is because the average-reward estimate was initialized to zero and it converged to around 0.5 corresponding to b_2 which skews the agent's state distribution towards the right. When the estimate passed the true value of 0.25, the RMSVE was quite low, however, the estimate quickly climbed to 0.5, resulting in the peak in RMSVE. Eventually the value estimates settled to values corresponding to an average-reward estimate of around 0.5. In contrast, value-based centering learned an average-reward estimate of just over 0.2, resulting in a smoother learning curve. In both the off-policy problems, value-based reward centering resulted in a higher rate of learning compared to the uncentered version, for both values of γ.

Overall, we saw that reward centering can improve the rate of learning of discounted-

reward prediction algorithms such as TD-learning, especially for large discount factors. While the simple way to center rewards is quite effective, value-based reward centering is better suited for general off-policy problems. We now consider the control problem.

5.4 Case Study: Q-learning with Centering

In this section, we examine the effects of reward centering used alongside the Q-learning algorithm (Watkins & Dayan, 1992). In particular, we highlight important related work and analyze the convergence as well as the fixed point of tabular Q-learning with value-based reward centering. Beyond the theory, we empirically study the effects of the tabular, linear, and non-linear variants of Q-learning with reward centering on various control problems.

Theory

We begin by specifying how to use Q-learning with reward centering. Q-learning is one of the oldest and most prevalent control algorithms. One of the reasons for its prevalence is that it is an off-policy algorithm: in the tabular case, it is guaranteed to converge to the value function of optimal policy while collecting data from an arbitrary behavior policy—even a random policy. Given its off-policy nature, we augment Q-learning with value-based reward centering. Since we use tabular, linear, and non-linear versions of this algorithm, we present a general form of its updates. At each time step, given an observation, the agent converts it into a feature vector $\mathbf{x}_t \in \mathbb{R}^d$, selects an action A_t, observes the reward signal R_{t+1} and the next observation, which it converts in to $\mathbf{x}_{t+1}$, and so on. In the tabular case, $\mathbf{x}_t$ is a one-hot vector of the size of the state space; in the linear case, $\mathbf{x}_t$ may be a tile-coding representation; in the non-linear case, $\mathbf{x}_t$ is the output of the last non-linear layer of an artificial neural network. In each case, the agent linearly combines the feature vector with an action-specific weight vector $\mathbf{w}^a \in \mathbb{R}^d, \forall a$ to obtain the action-value estimate $\hat{q}$.

Q-learning with value-based reward centering

At time step t, with the knowledge of $(\mathbf{x}_t, A_t, R_{t+1}, \mathbf{x}_{t+1})$, update the average-reward estimate and the per-action weights as:

$$\mathbf{w}_{t+1}^{A_t} \doteq \mathbf{w}_t^{A_t} + \alpha_t \, \delta_t \, \nabla_{\mathbf{w}_t} \hat{q}(\mathbf{x}_t, A_t), \tag{5.9}$$

$$\bar{R}_{t+1} \doteq \bar{R}_t + \eta \, \alpha_t \, \delta_t, \tag{5.10}$$

$$\text{where,} \quad \delta_t \doteq R_{t+1} - \bar{R}_t + \gamma \max_a (\mathbf{w}_t^a)^\top \mathbf{x}_{t+1} - (\mathbf{w}_t^{A_t})^\top \mathbf{x}_t. \tag{5.11}$$

Since the average-reward update now involves the values and not just the rewards, we cannot analyze its convergence separately from that of the value estimates. Fortunately, we can leverage some recent work to show that Q-learning with value-based reward centering converges almost surely in the tabular case. For the formal theorem and its proof (done by my collaborator—Yi Wan), please refer to the paper. I state the informal theorem statement here and analyze the fixed point.

Theorem 5.1. *If the Markov chain induced by the stationary behavior policy is irreducible, and a per-state-action step size is reduced appropriately, tabular Q-learning with value-based reward centering (5.9–5.11) converges almost surely: $\bar{R}_t$ and Q_t converge to a particular solution $(\tilde{q}^\gamma, \bar{r})$ of the following Bellman equations:*

$$\tilde{q}^\gamma(s,a) = \sum_{s',r} p(s', r|s, a)\big(r - \bar{r} + \gamma \max_{a'} \tilde{q}^\gamma(s', a')\big). \tag{5.12}$$

The convergence proof is a consequence of important recent work by Devraj and Meyn (2021). They showed that subtracting a particular quantity from the rewards in Q-learning results in a significantly stronger bound on asymptotic covariance. Depending on the quantity that is subtracted, there is a whole family of Q-learning variants that they call *Relative Q-learning*.

In particular, the general Relative Q-learning algorithm updates its tabular esti-

mates $\tilde{Q}^\gamma : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ at time step t using $(S_t, A_t, R_{t+1}, S_{t+1})$ as (in our notation):

$$\tilde{Q}^\gamma_{t+1}(S_t, A_t) \doteq \tilde{Q}^\gamma_t(S_t, A_t) + \alpha_t \big[R_{t+1} - f(\tilde{Q}^\gamma_t) + \gamma \max_{a'} \tilde{Q}^\gamma_t(S_{t+1}, a') - \tilde{Q}^\gamma_t(S_t, A_t)\big], \tag{5.13}$$

where, $f(\tilde{Q}^\gamma_t) \doteq \kappa \sum_{s,a} \mu(s,a) \tilde{Q}^\gamma_t(s,a)$, $\kappa > 0$ is a scalar, and $\mu : \mathcal{S} \times \mathcal{A} \to [0,1]$ is a probability mass function.

Devraj and Meyn showed that tabular Relative Q-learning converges almost surely to the value estimates $\tilde{\mathbf{Q}}^\gamma_\infty = \mathbf{q}^\gamma_* - k/(1-\gamma)\mathbf{1}$, where $\tilde{\mathbf{Q}}^\gamma_\infty$ denotes the vector of asymptotic value estimates, $\mathbf{q}^\gamma_*$ denotes the discounted action-value function of the optimal policy π^*_γ corresponding to the discount factor γ,[4] and k depends on κ, μ and $\mathbf{q}^\gamma_*$. Recall that the standard (uncentered) discounted value function $\mathbf{q}^\gamma_*$ has a state–action-independent offset of $r(\pi^*_\gamma)/(1-\gamma)$. Relative Q-learning can remove $k/(1-\gamma)$ of it, which is very promising. Devraj and Meyn left the choice of μ and κ as open questions.

We show that Q-learning with value-based reward centering is a member of the large family of Relative Q-learning algorithms with particular choices of μ and κ such that k is very close to $r(\pi^*_\gamma)$. In particular, recall from Chapters 3 and 4 that updating both the average-reward and value estimates using the TD error results in:

$$\bar{R}_t - \bar{R}_0 = \eta \Big(\sum_{s,a} Q_t(s,a) - \sum_{s,a} Q_0(s,a) \Big).$$

Without loss of generality, we can assume $\bar{R}_0 = 0$ and $\mathbf{Q}_0 = \mathbf{0}$. As a result, $\bar{R}_t = \eta \sum_{s,a} \tilde{Q}^\gamma_t(s,a)$. We can then combine the updates (5.9–5.10) in the tabular case:

$$\tilde{Q}^\gamma_{t+1}(S_t, A_t) \doteq \tilde{Q}^\gamma_t(S_t, A_t) + \alpha_t \big(R_{t+1} - \eta \sum_{s,a} \tilde{Q}^\gamma_t(s,a) + \max_{a'} \tilde{Q}^\gamma_t(S_{t+1}, a') - \tilde{Q}^\gamma_t(S_t, A_t)\big). \tag{5.14}$$

Comparing (5.13) and (5.14), we can see that Q-learning with value-based reward

[4] Recall that in the tabular case, there are optimal policies corresponding to each discount factor in $[0,1)$, however, they may not maximize the average reward. Consider the case of $\gamma = 0$ for intuition. The optimal policy corresponding to $\gamma = 0$ does not in general maximize the total reward in an full-RL episodic problem or the average reward in a full-RL continuing problem.

centering is an instance of Relative Q-learning with:

$$\mu(s,a) = \frac{1}{|\mathcal{S}||\mathcal{A}|} \ \forall s,a, \quad \text{and} \quad \kappa = \eta|\mathcal{S}||\mathcal{A}|.$$

Devraj and Meyn's (2021) convergence result then applies. That is,

$$\begin{aligned}\tilde{\mathbf{Q}}_t^\gamma \to \tilde{\mathbf{Q}}_\infty^\gamma &\doteq \mathbf{q}_*^\gamma - \frac{\kappa}{1-\gamma+\kappa}\mu^\top \mathbf{q}_*^\gamma \mathbf{1} \\ &= \mathbf{q}_*^\gamma - \frac{\eta}{1-\gamma+\eta|\mathcal{S}||\mathcal{A}|}\sum_{s,a} q_*^\gamma(s,a)\mathbf{1}. \end{aligned} \tag{5.15}$$

And,

$$\begin{aligned}\bar{R}_t \to \bar{R}_\infty &\doteq \eta \sum_{s,a} q_*^\gamma(s,a) - \frac{\eta^2|\mathcal{S}||\mathcal{A}|}{1-\gamma+\eta|\mathcal{S}||\mathcal{A}|}\sum_{s,a} q_*^\gamma(s,a) \\ &= \frac{\eta(1-\gamma)}{1-\gamma+\eta|\mathcal{S}||\mathcal{A}|}\sum_{s,a} q_*^\gamma(s,a). \end{aligned} \tag{5.16}$$

We now verify that $(\tilde{\mathbf{Q}}_\infty^\gamma, \bar{R}_\infty)$ satisfy the Bellman equations (5.12). Recall that the solutions of the Bellman equation are of the form $\big(\tilde{\mathbf{q}}_*^\gamma + \frac{k}{1-\gamma}\mathbf{1}, r(\pi_\gamma^*) - k\big)$. Since $\tilde{\mathbf{q}}_*^\gamma = \mathbf{q}_*^\gamma - \frac{r(\pi_\gamma^*)}{1-\gamma}$, we can re-write the solution class in terms of the discounted value function: $\big(\mathbf{q}_*^\gamma + \frac{(k-r(\pi_\gamma^*))}{1-\gamma}\mathbf{1}, r(\pi_\gamma^*) - k\big)$, or $\big(\mathbf{q}_*^\gamma - \frac{d}{1-\gamma}\mathbf{1}, d\big)$. For $d = \frac{\eta(1-\gamma)}{1-\gamma+\eta|\mathcal{S}||\mathcal{A}|}\sum_{s,a} q_*^\gamma(s,a)$, we can see that $(\tilde{\mathbf{Q}}_\infty^\gamma, \bar{R}_\infty)$ is a solution tuple of the Bellman equations.

We can now characterize how close $\bar{R}_\infty$ is to $r(\pi_\gamma^*)$. In general the expression for $\bar{R}_\infty$ (5.16) is cryptic. However, a special case can shed some light. We know that the average of the discounted value function for a policy w.r.t. that policy's steady-state distribution is: $\sum_{s,a} d_\pi(s,a) q_\pi^\gamma(s,a) = \frac{r(\pi)}{1-\gamma}$. Now suppose the steady-state distribution over state–action pairs is constant—$1/(|\mathcal{S}||\mathcal{A}|), \forall s,a$. For that policy, $\frac{1}{|\mathcal{S}||\mathcal{A}|}\sum_{s,a} q_\pi^\gamma(s,a) = \frac{r(\pi)}{1-\gamma}$. Substituting this in (5.16), we get:

$$\bar{R}_\infty = \frac{\eta|\mathcal{S}||\mathcal{A}|}{1-\gamma+\eta|\mathcal{S}||\mathcal{A}|} r(\pi_\gamma^*). \tag{5.17}$$

We can see that $\bar{R}_\infty$ approaches the true reward rate from below when $\eta|\mathcal{S}||\mathcal{A}| >> 1-\gamma$. We have to keep in mind, though, that this insight comes from a special case. More generally, we cannot concretely say where $\bar{R}_\infty$ (and hence $\tilde{\mathbf{Q}}_\infty^\gamma$) converges to. This is a shortcoming we wish to resolve in future work. However, this relation can serve as a rule of thumb.

Separately, Schneckenreither (2020) realized the Laurent series decomposition (5.2) suggests that an explicit estimate of the average reward can completely remove the offset. So they proposed an algorithm to estimate and subtract the average reward, with two important differences: (a) the average-reward estimate is updated only after non-exploratory actions, and (b) the algorithm has two discount factors to aim for the strongest optimality criterion—Blackwell optimality. Schneckenreither did not provide any convergence result for their algorithm. However, they analyzed that *if* the algorithm converged to the desired fixed point, then the resulting policy would be (Blackwell-)optimal. In our paper (Wan, Naik, & Sutton, 2021a), we showed that the average-reward estimate can be updated at every time step, including ones with exploratory actions, and showed almost-sure convergence of such algorithms. Combining those insights with Devraj and Meyn's, we could show the convergence of Q-learning with value-based reward centering.

Experiments

Note: If $\gamma = 1$, then Q-learning with value-based reward centering (5.9–5.11) is exactly Differential Q-learning (3.19–3.21). If $\gamma \in [0, 1)$, $\eta = 0$, and the average-reward estimate is initialized to zero, then we get back the standard Q-learning algorithm (Watkins & Dayan, 1992).

In this section, we present results on a set of control problems with tabular, linear, and non-linear function approximation to assess the benefits of reward centering. Most of the problems are included in Google DeepMind's CSuite (Zhao et al. 2022), which I have modified and added to at github.com/abhisheknaik96/csuite. We provide high-level descriptions here; see the repository documentation for all the details.

We begin with the Access-Control Queuing domain (Sutton & Barto, 2018: Chapter 10). This continuing problem was used in Chapter 3 as well and we repeat the high-level details here. The agent controls a server queue, where, at each time step, a job arrives at the front of the queue with one of four priorities $\{1, 2, 4, 8\}$. The

agent has to decide whether to accept or reject the job based on the number of free servers left (out of 10). If accepted, the agent gets a positive reward equal to the job's priority; if rejected, the job is removed from the queue and the agent gets zero reward. Occupied servers get free with a probability 0.06 at each time step. At each time step, the agent can observe the number of servers that are currently free and the priority of the job at the front of the queue.

We applied both the standard discounted Q-learning algorithm with and without centering on this problem, each for 50 independent runs of 50,000 steps. We tested various discount factors and step sizes as well as different values of the additional step-size parameter η for centering. All the learnable parameters were initialized to zero and both algorithms used an ϵ-greedy behavior policy with a fixed value of $\epsilon = 0.1$. While we are using discounted-reward algorithms, we are interested in the sum of undiscounted rewards obtained by the agents. For continuing problems, we consider the average reward obtained by the agents over a moving window of time steps. In all the plots in this section, the error bars or the shaded region indicates one standard error. The complete experimental details (for all the experiments in this section, particularly the exact values of the parameters tested) are in Appendix B.

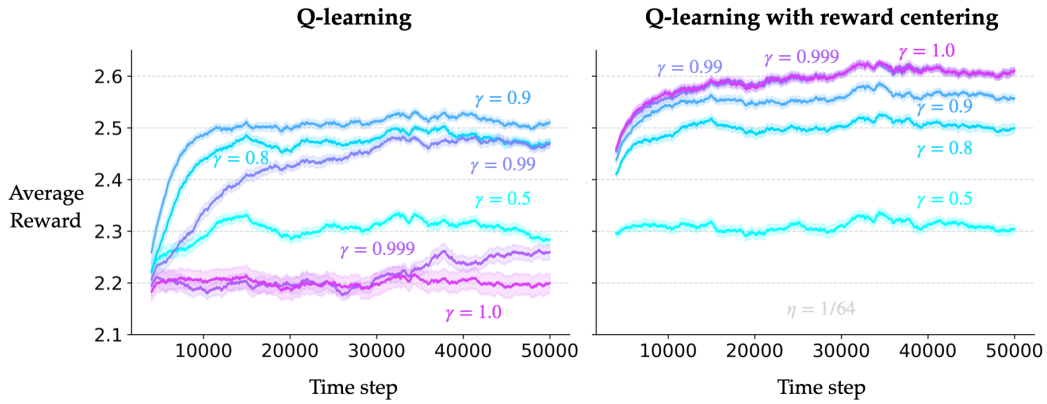


Figure 5.4: Learning curves corresponding to a range of discount factors for Q-learning with and without centering on the Access-Control Queuing problem. The x-axis denotes the number of agent-environment interactions and the y-axis denotes the rate of reward obtained by the agent over a moving window. The shaded region denotes one standard error. More details in-text.

Figure 5.4 shows the online performance for both algorithms (there is no separate testing period). For Q-learning without centering, the curves correspond to the step-size parameters that resulted in the fastest learning over the training period (quantified by the area under the learning curve); with centering, they correspond to the best step-size parameters for a fixed value of η (shown in grey in the figure). This does not always mean the best (α, η) pair for Q-learning with centering but that is okay since the results were robust to the choice of η. Throughout this section we followed this same practice of picking hyperparameters to plot learning curves.

We saw that the performance without centering first improved as the discount factor increased, then degraded; with centering, the performance did not degrade when the discount factor was close to one. For each discount factor, the performance with centering matched or exceeded that without centering.

To verify if centering indeed helped remove the potentially large state-independent term, we checked the magnitude of the learned values. One way is to compute the average value across all state-action pairs. However, this approach would typically lead to a poor approximation of the magnitude of learned values because many states (especially ones with low true values) may not occur frequently in the agent's ϵ-greedy interactions with the environment and hence their estimated values may stay close to their initialization. Instead, we checked the values of the states that actually occur in the agent's stream of experience, in particular the maximum action value (used to choose the argmax action) of the last 10% states that occurred during training. Table 5.1 shows these values for the parameters that resulted in Figure 5.4's learning

Table 5.1: Magnitude of learned values in Access-Control Queuing

Discount factor γ	**Q-learning**	**Q-learning with centering**
0.5	4.78	0.17
0.8	12.95	0.17
0.9	26.57	0.12
0.99	267.91	0.42
0.999	1434.47	0.51

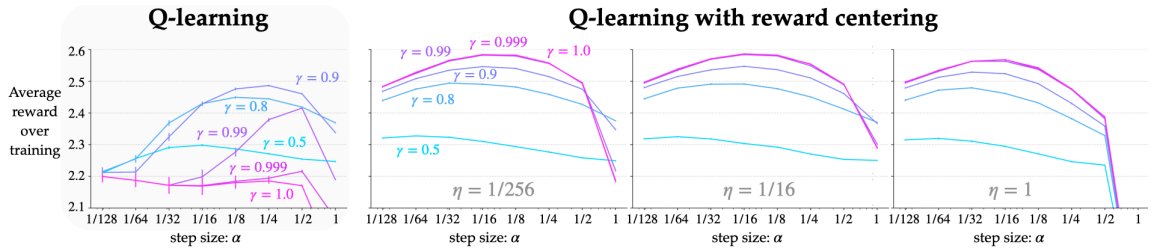


Figure 5.5: Parameter studies showing the sensitivity of the two algorithms' performance to their parameters on the Access-Control problem. The error bars indicate one standard error, which at times is less than the width of the lines. *Far left:* Without centering, rate of learning deteriorated with large discount factors for a broad range of the step-size parameter α. *Center to right:* For each discount factor, the performance with centering was better across a broad range of α. Moreover, the performance was robust across a large range of its second parameter η.

curves. As γ increased, the magnitude of learned values increased sharply without centering but remained small with centering.

These trends are quite general across the range of parameter values tested. Figure 5.5 shows the performance sensitivity to the methods' parameters. In particular, the x-axis denotes the step-size parameter α and the y-axis denotes the average reward obtained during the entire training period (which reflects the rate of learning). For both methods, the different curves correspond to different discount factors. The three plots on the right correspond to different values of the centering step-size parameter η. We saw the performance of Q-learning without centering deteriorated with large discount factors for a broad range of the step-size parameter α. In contrast, the performance with centering did not degrade; in fact, it improved all the way till $\gamma = 1$ for a wide range of η values. In addition, its performance was not sensitive to the choice of η.

We also observed the rate of learning of standard algorithms is significantly affected by a constant shift in the *problems'* rewards. Note that adding a constant to all the rewards does not change the ordering of the policies according to the total-reward or the average-reward criterion in continuing problems. Figure 5.6 shows the behavior of Q-learning with and without centering when applied to five problem variants with

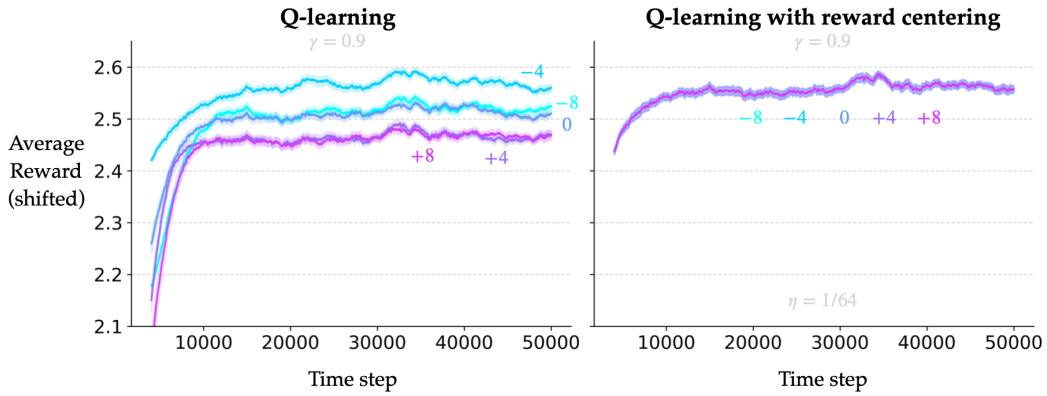


Figure 5.6: Learning curves with and without centering on slight variants of the Access-Control Queuing problem in which all the rewards shifted by a constant integer. The y-axis is shifted to compare learning curves for all the variants on the same scale. More details in-text.

one of $\{-8, -4, 0, 4, 8\}$ added to all the rewards. To compare the resulting rate of rewards across the problems, the plots are shifted post-hoc (so for instance, for the agent that operated in the problem variant that had rewards shifted by 8, the same number is subtracted from all the rewards that the agent obtained before plotting). The behavior of Q-learning without centering was substantially different on all the problem variants. Reward centering, unsurprisingly, results in similar behavior across the variants; we verified that the average-reward estimate indeed learns the average reward for every variant quickly. Figure 5.7 shows that these trends were also consistent across values of the step-size parameters. The plots are corresponding to $\gamma = 0.9$;

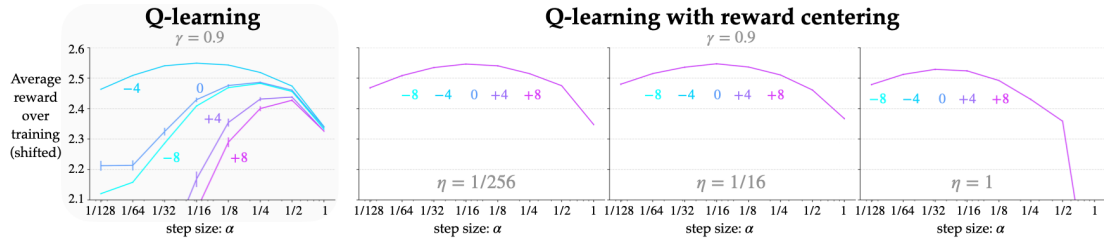


Figure 5.7: Parameter studies showing the performance sensitivity of Q-learning with and without centering to variants of the Access-Control problem. *Far left:* The performance of standard Q-learning's differed significantly on the different variants over a broad range of the step-size parameter α. *Center to right:* With centering, the performance was about the same across the problem variants, and was quite robust to the choice of its parameter η.

the trends did not differ much for different values of γ.

The next experiment was with PuckWorld and linear function approximation. In PuckWorld, the agent controls a puck-like object in a square rink where goal positions occur randomly. The agent can push the puck in any of the four cardinal directions. Repeated actions in a direction gives the puck some velocity that is upper-bounded due to friction. The agent observes six real numbers at each time step—the puck's position and velocity and the goal position in x and y directions—and gets a reward proportional to the negative distance to the goal. The goal position moves to a new random location every 300 time steps. The best policy takes the puck to the goal position as soon as possible.

We trained linear function approximators on this problem by tile-coding the 6-dimensional observation vector with 32 tilings of 4 tiles in each dimension. Each experiment was repeated for 20 runs of 300,000 steps each. We tested a range of step-size parameters and discount factors for both algorithms, which started from zero initializations of the weights and the scalar average-reward estimate. The behavior policy was ϵ-greedy with $\epsilon = 0.1$.

The trends were similar to but more dramatic than the previous tabular experi-

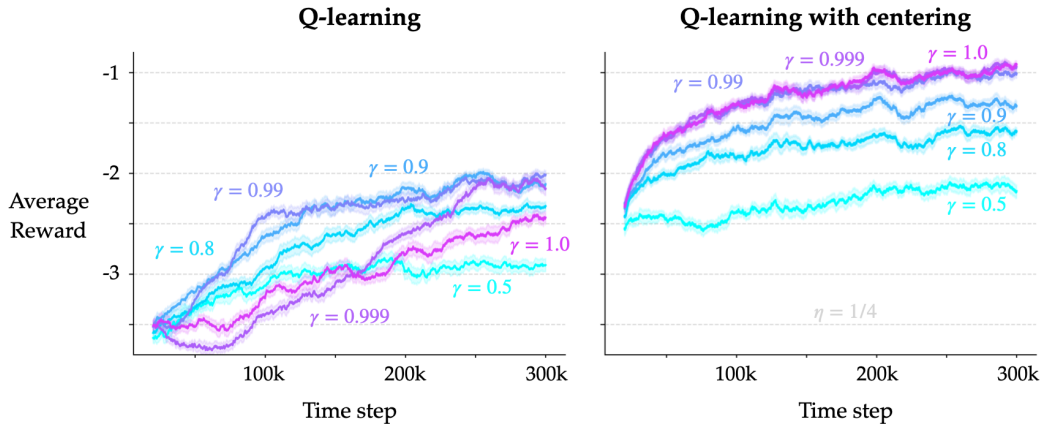


Figure 5.8: Learning curves corresponding to a range of discount factors for Q-learning with and without centering on the PuckWorld problem. With centering, the rate of learning of Q-learning was higher for each discount factor and did not degrade as $\gamma \to 1$.

ment. Figure 5.8 shows learning curves corresponding to the step-size parameter that resulted in the best performance for Q-learning without centering and the best step-size parameter for a fixed value of η for Q-learning with centering (the trends were consistent across the values of η tested; see the sensitivity plots in Appendix B). The x-axis denotes the number of agent-environment interactions while the y-axis denotes the rate of reward obtained by the agent over a moving window. The performance of Q-learning without centering suffered as γ increased; with centering, it did not. In fact, Q-learning with reward centering resulted in a better policy in shorter time for discount factors all the way to 1. The higher starting points of the learning curves also indicate that reward centering led to faster rate of learning with each of the discount factors tested.

Additionally, we observed similar trends when the rewards in the problem were shifted by a constant (Figure 5.9): Q-learning's rate of learning was highly sensitive to the shift, whereas with centering it was virtually unaffected. These trends were consistent across values of γ. We also found that the performance of Q-learning with centering was robust to a large range of its two parameters (sensitivity plots are in Appendix B).

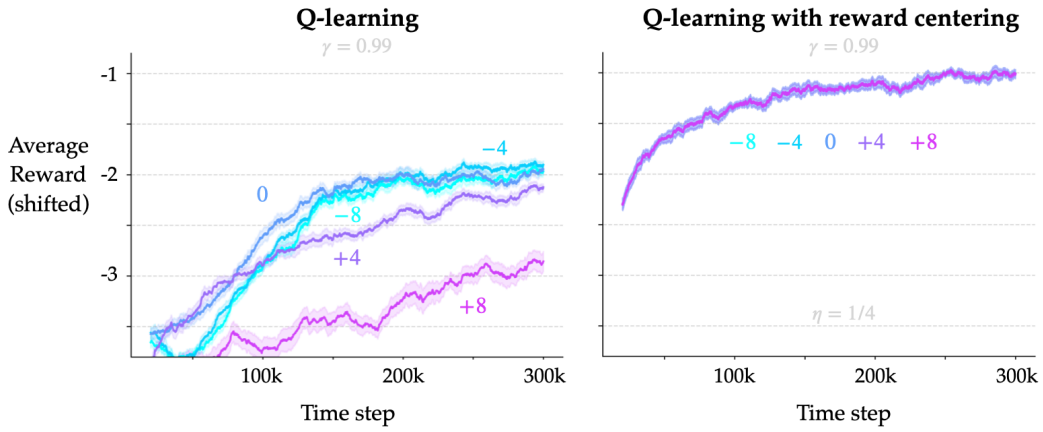


Figure 5.9: Learning curves for Q-learning with and without reward centering corresponding to $\gamma = 0.99$ on variants of the PuckWorld problem. The performance of Q-learning without centering was different on each variant while that with centering was roughly the same. Reward centering also resulted in relatively faster learning.

The results with non-linear function approximation on the Pendulum domain followed the same trends when we tested DQN (Mnih et al., 2015) and DQN with value-based reward centering. The agent controls the torque at the base of a one-link pendulum and gets a reward at each time step proportional to the negative angular distance of the pendulum from the upright position. The pendulum starts at rest pointing down. The agent can only apply a discrete amount of torque of $\{-1, 0, 1\}$ unit at each time step after observing three real numbers: the sine and cosine of the pendulum's angle w.r.t. pointing downwards, and the pendulum's angular velocity. There are no resets or timeouts; the agent must learn to keep the pendulum in the upright position. The pendulum repeatedly falls because the upright position is an unstable equilibrium and any exploratory actions can upset the pendulum.

We tested DQN with and without centering to estimate the action values in this problem. The artificial neural networks had two hidden layers with 64 units each with tanh activation functions, with the networks' weights trained using the Adam optimizer (Kingma & Ba, 2015) and the semi-gradient mean-squared-error loss. The weights were initialized in the standard way to small values around zero and the average-reward estimate was initialized to zero. The agents followed an ϵ-greedy behavior policy with $\epsilon = 0.1$ without annealing. Each experiment was run for 100,000 steps and repeated 15 times. See Appendix B for the rest of the implementation details, including the various deep-learning parameters.

One important difference in the non-linear implementation compared to the tabular or linear variants is that the updates are no longer fully online. Experience is collected at every time step and added to a buffer. After every few time steps, a batch of transitions is sampled from the buffer. The numerous parameters of the artificial neural network are then updated using the gradients computed via backpropagation (Rumelhart et al., 1986). Notably, the scalar average-reward estimate is updated using the mean of the TD errors computed across the sampled batch. The frequency and magnitude of the updates to the average-reward estimate are hence significantly

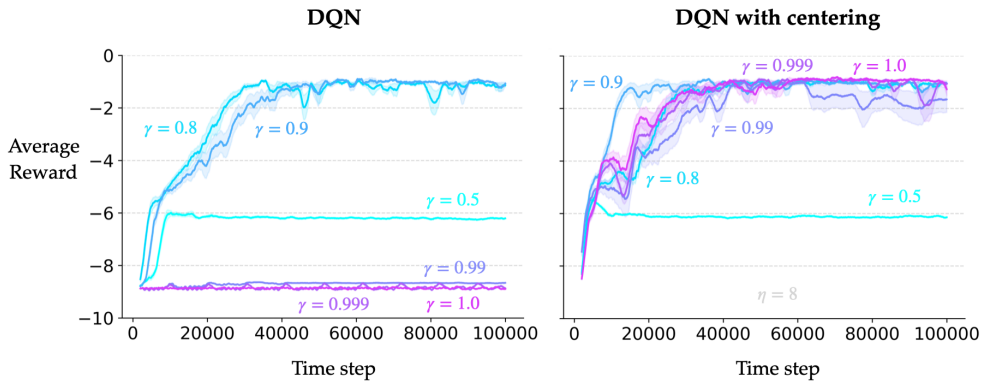


Figure 5.10: Learning curves corresponding to a range of discount factors for DQN with and without reward centering on the Pendulum problem. Notably, the rate of learning with centering did not degrade when $\gamma \to 1$. More details in-text.

lower than the fully online updates in tabular and linear implementations. As a result, the values of η tested for centering in the non-linear case are larger in comparison.

Figure 5.10 shows that the performance of DQN with and without centering corresponding to various discount factors. The curves correspond to step-size parameters that resulted in the best performance for DQN without centering and to those that resulted in the best performance for DQN with centering for a fixed value of η.

From the learning curves, we can infer that a discount factor of 0.5 was too small to solve the problem. The agents learned a good policy using DQN with discount factors 0.8 or 0.9 but failed to learn anything meaningful in 100k steps for discount factors 0.99 and larger. In contrast, with centering, the rate of learning did not degrade even with discount factors all the way up to 1.

We observed similar trends when the algorithms were tested on reward-shifted variants of the Pendulum problem. In Figure 5.11, we see that DQN's rate of learning was highly sensitive to the shift compared to when there was centering. We also found that the performance of DQN with centering was robust across a broad range of its parameters (see Appendix B).

We also performed a series of experiments with the Catch domain, using both linear and non-linear function approximators. In Catch, the agent controls a crate at the bottom row of a 2D pixel grid to catch falling fruits. The agent gets a +1 reward on

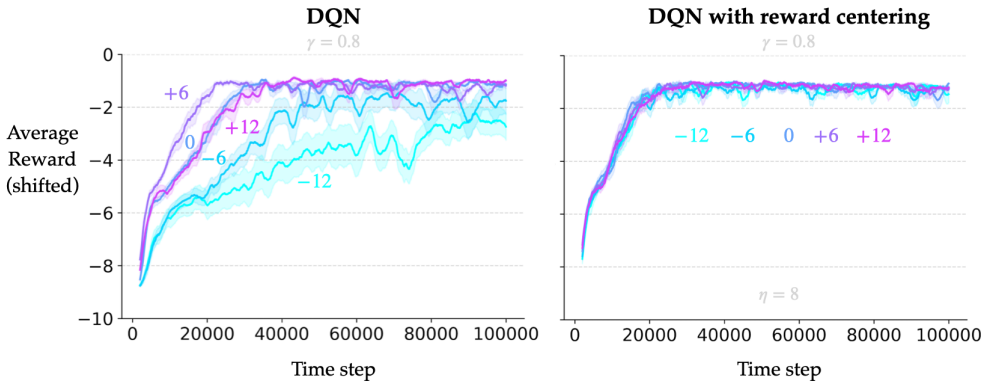


Figure 5.11: Learning curves for DQN with and without centering with $\gamma = 0.8$ on variants of the Pendulum problem. The performance of DQN without centering was different on each variant while that with centering was roughly the same.

successfully catching a fruit, -1 on dropping one, and 0 otherwise. At each time step, a new fruit is spawned with 10% probability in the top row, in a random column. More than one fruit may be falling at any point of time, and each fruit falls one pixel in one time step. The agent can choose among three actions: move the crate one pixel right, left, or stay put. There are two kinds of observation vectors available to an agent: a 3-dimensional real vector containing the x coordinate of the crate and the (x, y) coordinates of the lowermost fruit; a 50-dimensional binary vector which is the flattened version of the entire 10×5 pixel grid.

Figure 5.12 shows the learning curves of linear Q-learning with and without centering when applied to the Catch problem by tile-coding the 3D real-valued observation

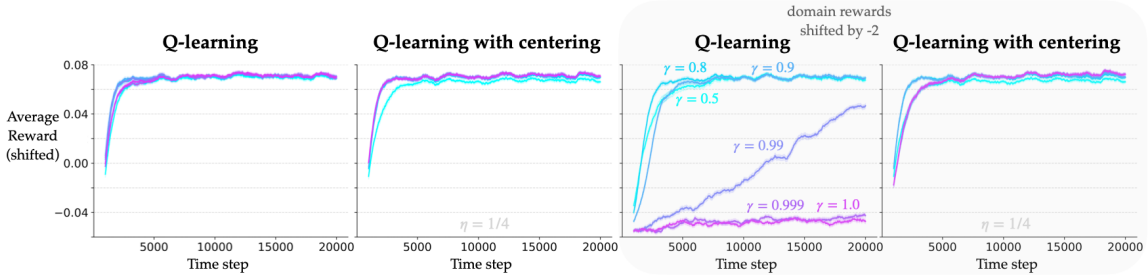


Figure 5.12: Learning curves for linear Q-learning with and without centering for various discount factors on two variants of Catch. *Left:* On the original problem, both algorithms performed well for all discount factors tested. *Right:* On the problem variant with all rewards shifted by -2, the rate of learning without centering was much lower for larger discount factors while that with centering was virtually unaffected.

vectors. The two plots on the left show learning curves on the original problem. In both cases, the learned policies were relatively good for all the discount factors tested, including a relatively low discount factor of 0.5. But as soon as the problem's rewards were shifted, the performance of Q-learning without centering suffered significantly for larger discount factors. The two plots on the right of Figure 5.12 show the performance on a variant of the Catch problem which had 2 subtracted from all the rewards. Recall that shifting the rewards by a constant does not change the ordering of the policies—the best policy remains unchanged. Without centering, the rate of learning was much lower for discount factors larger than 0.9 for all values of the step sizes tested; on the other hand, with centering, Q-learning continued to perform well.

These trends are further supported by the two plots on the left of Figure 5.13, where we show the sensitivity of the two algorithms to different problem variants. We created five problem variants by adding each of $\{-4, -2, 0, 2, 4\}$ to all the rewards one at a time. When the rewards are shifted by zero, we get the original problem; when the rewards are shifted by -2, we get the other problem variant discussed in Figure 5.12. On the x-axis is the effective step size for the linear function approximators and on the y-axis is the reward rate averaged over the entire training period. As before, the y-axis is adjusted to compare the performance on all the problem variants at the same scale. The plots in Figure 5.13 correspond to a discount factor of 0.8;

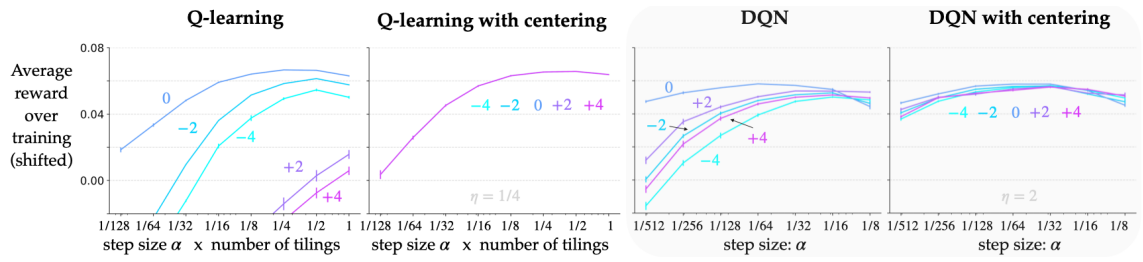


Figure 5.13: Parameter studies showing the sensitivity of the algorithms to their step-size parameter and to variants of the Catch problem. *Left:* Without centering, the rate of learning depended strongly on the problem variant and the step size, unlike that with centering. *Right:* The rate of learning of DQN with centering was roughly independent of the problem variant as well as the step-size parameter while that of DQN without centering depended on both.

the trends were similar for other discount factors. We found that the performance of Q-learning without centering was problem-dependent, whereas with centering the rate of learning was roughly the same regardless of the problem variant.

We observed this general trend across problems: if the problem rewards are roughly distributed around zero, then methods like Q-learning work quite well; if not, their rate of learning suffers. This makes sense in light of the Laurent-series decomposition.

The results with non-linear function approximation also exhibited the same trends. This time the agents observed the 50D binary vector and estimated the values of the three actions with networks having a hidden layer of 128 units. Each experiment was run for 80,000 steps and repeated 15 times, starting with the standard initialization of all the weights to small values around zero and the average-reward estimate to zero. The remaining agent details were same as those for Pendulum.

The two plots on the right of Figure 5.13 show that the rate of learning with standard DQN varied significantly across the range of step sizes and across the range of problem variants. In particular, the rate of learning was highest on the original problem for a broad range of the step sizes, although the best step size for each problem variant resulted in roughly the same rate of learning. With centering, the rate of learning was almost independent of the problem variant, that too for a broad range of the step size α.

5.5 Discussion and Conclusion

We saw a few positive trends across the set of experiments in the previous section:

- Reward centering can improve the performance of discounted methods for all discount factors, especially as $\gamma \to 1$.
- Reward centering can also make discounted methods robust to shifts in the problems' rewards.
- The parameter η for reward centering can be relatively easy to set.

However, there are some caveats to consider. Reward centering adds an additional element of non-stationarity to the learning problem because the average-reward estimate changes over time. As the result, the asymptotic variance of the updates is larger compared to when there is no centering (see, e.g., Figure 5.2).

The increased variance of updates also ties in to how the average reward is estimated. Ideally, the average reward is accurately estimated as quickly as possible and does not change too much per step. The simple centering technique quickly estimates the average reward but is affected by the per-step stochasticity. Value-based centering additionally relies on changing values and hence is relatively slower, but because it relies on the value estimates, per-step stochasticity in the problem dynamics may not affect it as much.[5] A step-size adaptation technique would be pertinent for estimating the average reward: larger step sizes when the errors are consistently large, smaller step sizes otherwise. Such an adaptation should be possible throughout the lifetime of the agent, since it may encounter new parts of the world having different kinds of reward signals.

Reward centering should be combined with approaches that deal with different *scales* of rewards. Reward centering makes learning algorithms robust to *shifts* in the rewards. In particular, it enables function approximators to focus on the essential differences between the state–actions rather than the differences in addition to an offset. However, the relative values themselves may have large magnitude. For instance, if all the rewards in a problem are multiplied by a constant, then the relative values will be scaled by the same constant. So it is pertinent to combine reward centering with *reward scaling*. The general idea of scaling quantities to a tractable range is quite common (e.g., van Hasselt et al., 2016; Pohlen et al., 2018). Building on this body of work, Schaul et al. (2021) recently proposed a lightweight trick that re-scales TD errors into an optimization-friendly range using problem-agnostic information in the

[5]Consider a fully deterministic tabular problem. The rewards are different at each step in general and so the simple centering technique always involves an error to correct. On the other hand, the per-step TD error would be zero in the value-based approach.

agent's data stream. With some care, their techniques can likely be extended from the episodic to the continuing case.

Another pertinent direction of work is to design a centering method that is guaranteed to estimate the average reward accurately even in the off-policy setting. We showed that value-based centering does not estimate the average reward exactly but its estimation can become increasingly accurate as the size of the problems' state–action space grows. However, an accurate estimation is appealing because it would completely remove the large state–action independent offset from the value estimates. The answer might be straightforward—like somehow constraining the value estimates to be mean zero which would constrain the $\bar{R}$ parameter to the accurate average-reward estimate—but has eluded me so far, much to my annoyance.

Finally, I would like to bring the reader's attention back to one of the most promising implications of centering: we can compute the approximate value function corresponding to *any* discount factor on the fly when we learn the average reward and the relative values separately. This opens the doors to an algorithm that adapts its discount factor over its lifetime. For instance, start with a small discount factor when there is a lot of uncertainty about the world, gradually increase it and adapt the relative values quickly, and perhaps reduce it again during another period of uncertainty.

Connections to other related approaches

The reward-centering idea is compatible with (and not a competitor of) approaches like reward scaling and advantage estimation. In addition, reward centering is a particular type of reward shaping. We briefly discuss these points now.

Dividing all the rewards with a (potentially changing) scalar number is typically referred to as reward scaling (see, e.g., Engstrom et al., 2020). Just like reward centering, reward scaling does not change the ordering of policies in a continuing problem. Scaling reduces the spread of the rewards, centering brings them close to zero, both of which can be favorable to complex function approximators such as artificial neural

networks that are used for value estimation starting from a close-to-zero initialization. The popular stable_baselines3 repository scales (and clips[6]) the rewards by a running estimate of the variance of the discounted returns (github.com/DLR-RM/stable-baselines3/blob/master/stable_baselines3/common/vec_env/vec_normalize.py#L256). Mean-centering the rewards as well would be beneficial for continuing domains.

Note that the mechanism of computing the mean and variance is trickier in the off-policy setting than the on-policy setting. Our TD-error-based technique is likely part of the final solution for the off-policy setting. Simply maintaining a running estimate of the variance (as in the stable_baselines' approach) introduces a bias. As mentioned earlier, Schaul et al.'s (2021) technique is a good starting point.

Reward centering and the advantage function have orthogonal benefits. The advantage function benefits the actor by reducing the variance of the updates in the policy space (Sutton & Barto, 2018; Schulman et al., 2016). On the other hand, reward centering benefits the critic's or baseline's estimation by eliminating the need to estimate the large state-independent constant offset. Both the quantities involved in the advantage function—$a_\pi^\gamma(s,a) = q_\pi^\gamma(s,a) - v_\pi^\gamma(s)\ \forall s, a$—have the large state-independent offset $r(\pi)/(1-\gamma)$. The net effect of the offset is zero when they are subtracted. But the key point is that both the state- and action-value estimates include the large offset. Reward centering removes the need to estimate the large offset for both the state- and action-value function, which simplifies the critic-estimation problem. The actor update is left unchanged with reward centering because the advantage function itself remains unchanged: $\tilde{a}_\pi^\gamma(s,a) = \tilde{q}_\pi^\gamma(s,a) - \tilde{v}_\pi^\gamma(s)$, because $\tilde{q}_\pi^\gamma(s,a) = q_\pi^\gamma(s,a) - r(\pi)/(1-\gamma)$ and $\tilde{v}_\pi^\gamma(s) = v_\pi^\gamma(s) - r(\pi)/(1-\gamma)$.

Hence, we expect reward centering to benefit all the algorithms that estimate values, which include all actor-critic methods that involve advantage estimation. An empirical study of centered variants of several common policy-based algorithms (like

[6]Reward clipping in general changes the problem. Blinding the agent from large rewards can impose a performance ceiling or make some games impossible to solve (Schaul et al.'s (2021) Section 4.3 discusses this in the context of Atari problems).

Schulman et al.'s (2015, 2017) TRPO and PPO) is a ripe avenue of future work.

Related to baselines is the idea of bias weight. In theory, if a bias weight (corresponding to a bias feature of one) is equal to the state–action-independent offset, then the rest of the function-approximation capacity can estimate the relative values. However, in our experiments we observed that the learned bias weight was rarely close to true value of the offset. Moreover, it is more desirable to separate $r(\pi)$ and scaling effect of $1/(1-\gamma)$. Along these lines, Tsitsiklis and Van Roy (2002) suggested setting a linear approximator's bias feature proportional to $1/(1-\gamma)$ and learning the corresponding bias weight. Reward centering takes this idea forward. Since we know what the corresponding bias weight is, we can have an explicit update rule for it (à la Sutton (1988b)), which can lead to faster learning. Besides, in the control problem, there is no need to estimate standard discounted values with the offset, as Tsitsiklis and Van Roy propose doing. Instead, via reward centering, we can simply estimate the centered (relative) values.

Reward centering can be seen as Ng et al.'s (1999) reward shaping with a constant state-independent potential function: $\Phi(s) = r(\pi)/(1-\gamma)\ \forall s$. Their Theorem 1 then reiterates that reward centering does not change the optimal policy of the problem. A possible drawback of reward shaping is that it can be tricky to fully specify the potential-based shaping function, especially for problems with large state spaces. In the case of reward centering, the specification is relatively easy: the potential function is constant across the entire state space, and we know how to learn the average reward reliably from data.

Conclusion

Reward centering is a simple idea with large benefits and should be used alongside all the discounted algorithms. When combined with appropriate step-size adaptation and reward-scaling techniques, I believe it will be a key enabler for agents to learn quickly and continually over their lifetimes.

Chapter 6

Conclusions and Future Work

In this concluding chapter of my dissertation, I summarize my contributions and outline my recommendations for future work.

6.1 Contributions

The primary contributions of my dissertation are:

1. In Chapter 3, I presented one-step learning methods for on- and off-policy prediction and control. The core idea is simple: estimate the average reward using the TD error. As a result, Differential TD-learning and Differential Q-learning are applicable in the off-policy setting and do not require any special reference function. Additionally, the performance of the algorithms is quite robust to their parameters, making the algorithms relatively easy to use.

2. In Chapter 4, I extended the one-step prediction algorithm to the case of multi-step updates using eligibility traces. In the on-policy setting, I showed convergence results with linear function approximation. I also presented the first convergence results in the off-policy setting for a family of multi-step tabular average-reward algorithms. I validated the theoretical results and intuitions through careful experiments.

3. In Chapter 5, I showed that ideas from the average-reward formulation can also improve solution methods for the discounted-reward formulation. In particu-

lar, learning the average reward and subtracting it from the observed rewards can improve the performance of discounted solution methods—especially as the discount factor approaches one—and also make them more robust to shifts in the problems' rewards. Through a series of experiments, I demonstrated the benefits of reward centering with tabular, linear, and non-linear function approximation.

The main ideas in this dissertation arise from first principles. From a rearrangement of the differential Bellman equations, we realized that a sample average of the TD error is an unbiased estimate of the average reward even in the off-policy setting. By making the changes to the average-reward estimate proportional to the changes in the value estimates, we showed the convergence of off-policy multi-step algorithms. From the Laurent-series decomposition of the discounted value function, we inferred that estimating the average reward separately can make the estimation of the discounted values easier.

The resulting algorithms and techniques are simple and practical. The average reward is estimated using information that the agent already has. Despite the differences in theory and analysis, the algorithms for average-reward prediction and control are very similar to the prevalent discounted-reward algorithms. We also found that the performance of average-reward algorithms and the methods for reward centering are quite robust to their parameters: practitioners can achieve good performance without spending a lot of time in tuning parameters.

Finally, all the contributions are applicable in the continuing setting, in which the agent-environment interaction goes on *ad infinitum*. Taken together, the contributions of this dissertation are clear steps based on first principles that advance the frontiers of RL research on continuing problems based on the average-reward formulation, and more generally towards developing simple and practical learning algorithms for long-lived agents.

Much work remains, however, to extend and combine this work with orthogonal advances in reinforcement learning for the grander ambitions of AI.

6.2 Directions of Future Work

In this section, I outline pertinent directions of future work that go beyond the individual topics of this dissertation (the future work specific to each contribution is discussed at the end of each chapter).

First of all, we need to systematically evaluate average-reward algorithms on a wide range of continuing problems to get a more accurate assessment of the algorithmic capabilities. The algorithms tested in this dissertation appear promising on several problems with tabular, linear, and non-linear function-approximation (recall that setting $\gamma = 1$ with reward centering results in an average-reward algorithm). However, the evaluation here is focused on teasing apart specific properties of the algorithms rather than testing them more generally. In particular, it is pertinent to test the average-reward algorithms in this dissertation as well as others on a variety of problems, preferably those that are not designed for RL research.

A wide-ranging empirical study requires a suite of continuing environments, which is currently lacking in the RL literature. In the episodic setting, collections of problems such as Gym (Brockman et al., 2016) and ALE (Bellemare et al., 2013) have accelerated research. Other popular libraries such as MuJoCo (Todorov et al., 2012), DeepMind Lab (Beattie et al., 2016), PyGame Learning Environment (PLE: Tasfi, 2016), PyBullet (Coumans & Bai, 2016), DeepMind Control Suite (Tassa et al., 2020), bsuite (Osband et al., 2020) are all primarily episodic, and have some naturally continuing problems that have been made episodic via timeouts. One exception is the Jelly Bean World (Platanios et al., 2020). The lack of a collection of continuing problems has limited research on the continuing setting (which further reduces the interest in developing and collecting a set of continuing domains—a vicious cycle). I started putting together continuing domains from the literature as well as new ones

(Naik et al., 2021). Some colleagues at DeepMind's former Alberta office then engineered an extensive infrastructure to make what we started calling *CSuite*'s interface as accessible and easy to contribute to as Gym's. Unfortunately, the office shut down after a preliminary release of CSuite with just a few illustrative problems (Zhao et al., 2022). I continued working on CSuite, but much work needs to be done. It would be great to get more help in creating a collection of problems that cover both illustrative domains as well as ambitious real-world problems.

An important set of baselines in an empirical study of average-reward algorithms are the equivalent discounted-reward algorithms. Discounted algorithms do not optimize the average-reward objective but are pertinent baselines nonetheless for continuing problems.[1] The billion-dollar question is: *for continuing problems, should we use average-reward algorithms or discounted-reward algorithms?* I really wanted to, but I did not answer this question during my Ph.D. Nevertheless, I made concrete progress towards answering it.

One of the things that I learned is that extra care needs to be taken when comparing average-reward and discounted-reward algorithms. For instance, I found that the range of the reward signal is a hidden parameter that can significantly influence the performance of standard discounted methods. When I was testing continuing variants of MinAtar problems (Young & Tian, 2019), I discovered through a bug in my code that a DQN agent failed to learn anything if it observed rewards that had a constant such as +1 added to all of them. On the other hand, the differential version of DQN performed almost the same in both cases. After replicating the issue in smaller problems, reward centering was conceived as an idea to make discounted methods robust to any constant shifts in the problems' rewards.

Having explored reward centering more deeply now, I have realized it has another intriguing implication. Learning both the average reward and the centered discounted

[1]The discounted-reward *problem* is not well-defined for continuing control with function approximation (Sutton & Barto, 2018: Section 10.4; Naik et al., 2019). However, discounting can be used as a parameter within *solution methods*.

estimates allows the agent to rapidly estimate the discounted value function corresponding to *any* discount factor. Concretely, consider the agent has estimated the average reward $\bar{R}$ and the centered discounted value function $\tilde{\mathbf{v}}^{\gamma_1}$ to some level of accuracy. With just this information, the agent can form an estimate of the standard discounted value function corresponding to another discount factor γ_2 via $\frac{\bar{R}}{(1-\gamma_2)} + \tilde{\mathbf{v}}^{\gamma_1}$. This is an estimate, of course, but it can be improved quickly with a few samples of experience—potentially with old experience from a buffer or a parameterized model. In contrast, with standard methods, it would take comparatively longer to raise the estimates to the new mean value and adapt the relative values. Hence, with reward centering, we can imagine efficient methods that adapt their discount factors over time: a low discount rate to learn quickly amidst a lot of uncertainty—like in the beginning of training—and when the world is more predictable, a higher discount rate to estimate the policy that maximizes the total amount of reward obtained by the agent.

The interest in letting the agent set the discount factor on its own is not new. Xu et al. (2018) started a line of work that changes the discount factor based on its effect on the evaluation criteria. Such gradient-based meta-learning approaches are quite general and have since been integrated with prevalent deep RL algorithms (e.g., Zahavy et al., 2020). On the other hand, specific knowledge about discounting can inspire specific ways of changing the discount factor. For instance, Dong et al. (2022) recently proposed a method to monotonically increase the discount factor over time (and showed it has bounded regret). Similarly, Khetarpal et al. (2023) proposed increasing the discount factor in the offline setting using a notion of model uncertainty that reduces over time. However, since the world may appear to be changing to any learning agent in the big-world perspective, it might be useful to reduce the discount factor, say, when the agent is uncertain about the part of the world that it is in. I would like to explore different ways of modeling uncertainty in RL (e.g., like Abbas (2020)). With reward centering, this could result in a technique that is faster than

an average-reward algorithm and more effective that a discounted algorithm with a fixed discount factor. I am excited to explore this idea which could be key to efficient long-lived learning systems.

There are also several directions of work to broaden the scope of the average-reward formulation for RL, such as with exploration. Exploration is especially important in continuing problems because—unlike in episodic problems—there are no resets (or timeouts). Agents that learn from a single stream of experience may get 'stuck' for long periods of time with simple exploration methods such as ϵ-greedy action selection that do not explicitly take the agents' experience into account (e.g., see Machado et al.'s (2020) RiverSwim experiment or Sharma et al.'s (2022) Figure 3). Mahadevan (1996) identified a related phenomenon: an agent using Schwartz's (1993) R-learning algorithm got stuck in "limit cycles". As a result, Mahadevan recommended using higher levels of exploration when using strategies such as ϵ-greedy action selection. This is a strong reason to pursue principled exploration methods (like Machado, 2019) beyond the prevalent dithering approaches, especially in the context of long-lived agents.

This dissertation considers value-based methods; *policy-based* methods form an important class of RL algorithms. The standard policy gradient (Sutton et al., 1999b) and the natural policy gradient (Kakade, 2001a) were both derived for the average-reward formulation. However, there are lots of recent developments in policy-based methods for the discounted-reward formulation that can be extended to the average-reward formulation. Zhang and Ross (2021) recently took an important step in this direction by analyzing the average-reward version of TRPO (Schulman, 2015). Interestingly, their experiments showed that average-reward TRPO significantly outperformed discounted TRPO on continuing variants of MuJoCo problems. Such extensions for other kinds of policy-based methods would help make average-reward algorithms applicable to a larger set of problems (particularly ones with continuous action spaces).

Finally, this dissertation deals exclusively with *model-free* learning approaches. There is a rich body of RL literature that involves the agent learning a *model* of the environment's transition and reward dynamics, which it can then use to improve its value estimates or its policy without additional interactions with the environment (e.g., Sutton, 1990). There are several aspects of planning that work as is in the average-reward formulation, but others need more care. For example, the planning variants of all the differential-style learning algorithms are straightforward and are analyzed in our one-step paper (Wan, Naik, & Sutton, 2021a). On the other hand, the chain structure of the problem is relevant in average-reward planning: MDPs with more than one chain—multichain MDPs—require average-reward estimates per state and hence the planning algorithms are more involved (e.g., see Puterman's (1994) Chapter 9). Such ideas need to be fleshed out to integrate average-reward planning into a complete learning agent that learns to achieve goals throughout its lifetime (e.g., Sutton, 2022).

References

Abbas, Z. (2020). *Selective Dyna-style Planning Using Neural Network Models with Limited Capacity*. M.Sc. thesis, University of Alberta.

Abounadi, J., Bertsekas, D., & Borkar, V. S. (1998). *Stochastic Approximation for Nonexpansive Maps: Application to Q-Learning*, Report LIDS-P-2433, Laboratory for Information and Decision Systems, MIT.

Abounadi, J., Bertsekas, D., & Borkar, V. S. (2001). Learning Algorithms for Markov Decision Processes with Average Cost. *SIAM Journal on Control and Optimization*.

Abbasi-Yadkori, Y., Bartlett, P., Bhatia, K., Lazic, N., Szepesvári, C., & Weisz, G. (2019). POLITEX: Regret Bounds for Policy Iteration Using Expert Prediction. *International Conference on Machine Learning*.

Auer, R., & Ortner, P. (2006). Logarithmic Online Regret Bounds for Undiscounted Reinforcement Learning. *Advances in Neural Information Processing Systems*.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., Legg, S., & Petersen, S. (2016). DeepMind Lab. *ArXiv:1612.03801*.

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*.

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., & Wang, Z. (2020). Autonomous Navigation of Stratospheric Balloons Using Reinforcement Learning. *Nature*.

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

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic Programming*. Athena Scientific.

Blackwell, D. (1962). Discrete Dynamic Programming. *The Annals of Mathematical Statistics.*

Borkar, V. S. (2009). *Stochastic Approximation: A Dynamical Systems Viewpoint.* Springer.

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX — A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research.*

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *ArXiv:1606.01540.*

Coumans, E., & Bai, Y. (2016). PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning. pybullet.org

Das, T. K., Gosavi, A., Mahadevan, S., & Marchalleck, N. (1999). Solving Semi-Markov Decision Problems Using Average Reward Reinforcement Learning. *Management Science.*

Devraj, A., & Meyn, S. (2021). Q-learning With Uniformly Bounded Variance. *IEEE Transactions on Automatic Control.* Also *ArXiv:2002.10301.*

Dewanto, V., Dunn, G., Eshragh, A., Gallagher, M., & Roosta, F. (2020). Average-Reward Model-Free Reinforcement Learning: A Systematic Review and Literature Mapping. *ArXiv:2010.08920.*

Dong, S., Van Roy, B., & Zhou, Z. (2022). Simple Agent, Complex Environment: Efficient Reinforcement Learning with Agent State. *Journal of Machine Learning Research.*

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., & Madry, A. (2019). Implementation Matters in Deep RL: A Case Study on PPO and TRPO. *International Conference on Learning Representations.*

Even-Dar, E., Mansour, Y., & Bartlett, P. (2003). Learning Rates for Q-learning. *Journal of Machine Learning Research.*

Fedus, W., Gelada, C., Bengio, Y., Bellemare, M. G., & Larochelle, H. (2019). Hyperbolic Discounting and Learning Over Multiple Horizons. *ArXiv:1902.06865.*

Gelada, C., & Bellemare, M. G. (2019). Off-Policy Deep Reinforcement Learning by Bootstrapping the Covariate Shift. *AAAI Conference on Artificial Intelligence.*

Gosavi, A. (2004). Reinforcement Learning for Long-Run Average Cost. *European Journal of Operational Research.*

Grand-Clément, J., & Petrik, M. (2023). Reducing Blackwell and Average Optimality to Discounted MDPs via the Blackwell Discount Factor. *ArXiv: 2302.00036*.

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

Hallak, A., & Mannor, S. (2017). Consistent On-Line Off-Policy Evaluation. *International Conference on Machine Learning*.

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

Jaksch, T., Ortner, R., & Auer, P. (2010). Near-Optimal Regret Bounds for Reinforcement Learning. *Journal of Machine Learning Research*.

Jalali, A., & Ferguson, M. J. (1989). Computationally Efficient Adaptive Control Algorithms for Markov Chains. *IEEE Conference on Decision and Control*.

Jalali, A., & Ferguson, M. J. (1990). A Distributed Ssynchronous Algorithm for Expected Average Cost Dynamic Programming. *IEEE Conference on Decision and Control*.

Kakade, S. M. (2001a). A Natural Policy Gradient. *Advances in Neural Information Processing Systems*.

Kakade, S. M. (2001b). Optimizing Average Reward Dsing Discounted Rewards. *International Conference on Computational Learning Theory*.

Kaufmann, E., Bauersfeld, L., Loquercio, A., Müller, M., Koltun, V. & Scaramuzza, D. (2023). Champion-Level Drone Racing Using Deep Reinforcement Learning. *Nature*.

Kearns, M., & Singh, S. (2002). Near-Pptimal Reinforcement Learning in Polynomial Time. *Machine Learning*.

Khalil, H. K. (2002). *Nonlinear Systems*. Prentice Hall.

Khetarpal, K., Vernade, C., O'Donoghue, B., Singh, S., & Zahavy, T. (2023). POMRL: No-Regret Learning-to-Plan with Increasing Horizons. *Transactions in Machine Learning Research*.

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *ArXiv:1412.6980*.

Konda, V. R., & Tsitsiklis, J. N. (1999). Actor-Critic Algorithms. *Advances in Neural Information Processing Systems*.

Konda, V. R. (2002). *Actor-Critic Algorithms*. Ph.D. dissertation, MIT.

Levin, D. A., & Peres, Y. (2017). *Markov Chains and Mixing Times*. American Mathematical Society.

Liu, Q., Li, L., Tang, Z., & Zhou, D. (2018). Breaking the Curse of Horizon: Infinite-Horizon Off-Policy Estimation. *Advances in Neural Information Processing Systems*.

Machado, M. C. (2019). *Efficient Exploration in Reinforcement Learning through Time-Based Representations*. Ph.D. dissertation, University of Alberta.

Machado, M. C., Bellemare, M. G., & Bowling, M. (2020). Count-Based Exploration with the Successor Representation. *AAAI Conference on Artificial Intelligence*.

Maei, H. R., Szepesvári, C., Bhatnagar, S., & Sutton, R. S. (2010). Toward Off-Policy Learning Control with Function Approximation. *International Conference on Machine Learning*.

Maei, H. R. (2011). *Gradient Temporal-Difference Learning Algorithms.* Ph.D. dissertation, University of Alberta.

Mahadevan, S. (1996). Average Reward Reinforcement Learning: Foundations, Algorithms, and Empirical Results. *Machine Learning*.

Marbach, P., & Tsitsiklis, J. N. (2001). Simulation-Based Optimization of Markov Reward Processes. *IEEE Transactions on Automatic Control*.

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*.

Mousavi, A., Li, L., Liu, Q., & Zhou, D. (2020). Black-Box Off-Policy Estimation for Infinite-Horizon Reinforcement Learning. *International Conference on Learning Representations*.

Naik, A., Shariff, R., Yasui, N., Yao, H., & Sutton, R. S. (2019). Discounted Reinforcement Learning Is Not an Optimization Problem. *Optimization Foundations for Reinforcement Learning Workshop at the Conference on Neural Information Processing Systems*. Also *ArXiv:1910.02140*.

Naik, A., Abbas, Z., White, A., & Sutton, R. S. (2021). Towards Reinforcement Learning in the Continuing Setting. *Never-Ending Reinforcement Learning workshop at the International Conference on Learning Representations*.

Naik, A., & Sutton, R. S. (2022). Multi-Step Average-Reward Prediction via Differential TD(λ). *Conference on Reinforcement Learning and Decision Making*.

Ng, A., Harada, D., & Russell, S. (1999). Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. *International Conference on Machine Learning*.

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvári, Singh, S., Van Roy, B., Sutton, R. S., Silver, D., & van Hasselt, H. (2020). Behaviour Suite for Reinforcement Learning. *International Conference on Learning Representations*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*.

Pohlen, T., Piot, B., Hester, T., Azar, M. G., Horgan, D., Budden, D., Barth-Maron, G., van Hasselt, H., Quan, J., Večerík, M., Hessel, M., Munos, R., & Pietquin, O. (2018). Observe and Look Further: Achieving Consistent Performance on Atari. *ArXiv:1805.11593*.

Platanios, E. A., Saparov, A., & Mitchell, T. (2020). Jelly Bean World: A Testbed for Never-Ending Learning. *International Conference on Learning Representations*.

Precup, D., Sutton, R. S., & Singh, S. (2000). Eligibility Traces for Off-Policy Policy Evaluation. *International Conference on Machine Learning*.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Qu, G., & Wierman, A. (2020). Finite-Time Analysis of Asynchronous Stochastic Approximation and Q-learning. *Conference on Learning Theory.*

Ren, Z., & Krogh, B. H. (2001). Adaptive Control of Markov Chains with Average Cost. *IEEE Transactions on Automatic Control*.

Rubinstein, R. (1981). *Simulation and the Monte Carlo Method*. John Wiley & Sons.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Representations by Back-Propagating Errors. *Nature*.

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning Using Connectionist Systems.* Technical Report, Engineering Department, Cambridge University.

Schaul, T., Ostrovski, G., Kemaev, I., & Borsa, D. (2021). Return-Based Scaling: Yet Another Normalisation Trick for Deep RL. *ArXiv:2105.05347*.

Schneckenreither, M. (2020). Average Reward Adjusted Discounted Reinforcement Learning: Near-Blackwell-Optimal Policies for Real-World Applications. *ArXiv:2004.00857*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust Region Policy Optimization. *International Conference on Machine Learning*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2016). High-Dimensional Continuous Control Using Generalized Advantage Estimation. *International Conference on Learning Representations*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *ArXiv:1707.06347*.

Schwartz, A. (1993). A Reinforcement Learning Method for Maximizing Undiscounted Rewards. *International Conference on Machine Learning*.

Sharma, A., Xu, K., Sardana, N., Gupta, A., Hausman, K., Levine, S., & Finn, C. (2021). Autonomous Reinforcement Learning: Formalism and Benchmarking. *International Conference on Learning Representations*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go Through Self-Play. *Science*.

Singh, S. (1994). Reinforcement Learning Algorithms for Average-Payoff Markovian Decision Processes. *AAAI Conference on Artificial Intelligence*.

Singh, S., Jaakkola, T., & Jordan, M. (1994). Learning Without State-Estimation in Partially Observable Markovian Decision Processes. *Machine Learning Proceedings*.

Sutton, R. S. (1988a). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*.

Sutton, R. S. (1988b). *NADALINE: A Normalized Adaptive Linear Element That Learns Efficiently*. GTE Laboratories Technical Report.

Sutton, R. S. (1990). Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. *International Conference on Machine Learning*.

Sutton, R. S., Precup, D., & Singh, S. (1999a). Between MDPs and Semi-MDPs: A

Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*.

Sutton, R. S., McAllester, D. A., Singh, S., & Mansour, Y. (1999b). Policy Gradient Methods for Reinforcement Learning With Function Approximation. *Advances in Neural Information Processing Systems*.

Sutton, R. S., Koop, A., & Silver, D. (2007). On the Role of Tracking in Stationary Environments. *International Conference on Machine Learning*.

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. *International Conference on Autonomous Agents and Multiagent Systems*.

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*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* MIT Press.

Sutton, R. S. (2022). The Quest for a Common Model of the Intelligent Decision Maker. *Conference on Reinforcement Learning and Decision Making*. Also *ArXiv:2202.13252*.

Sutton, R. S., Bowling, M., & Pilarski, P. M. (2022). The Alberta Plan for AI Research. *ArXiv:2208.11173*.

Tang, Z., Feng, Y., Li, L., Zhou, D., & Liu, Q. (2019). Doubly Robust Bias Reduction in Infinite Horizon Off-Policy Estimation. *International Conference on Learning Representations*.

Tasfi, N. (2016). PyGame Learning Environment, *Github:*ntasfi/PyGame-Learning-Environment

Tassa, Y., Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., & Heess, N. (2020). DeepMind Control Suite. *ArXiv:2006.12983*.

Todorov, E., Erez, T., & Tassa, Y. (2012). MuJoCo: A Physics Engine for Model-Based Control. *International Conference on Robotics and Automation*.

Tsitsiklis, J. N., & Van Roy, B. (1997). An Analysis of Temporal-Difference Learning With Function Approximation. *IEEE Transactions on Automatic Control*.

Tsitsiklis, J. N., & Van Roy, B. (1999). Average Cost Temporal-Difference Learning.

Automatica.

Tsitsiklis, J. N., Van Roy, B. (2002). On Average Versus Discounted Reward Temporal-Difference Learning. *Machine Learning*.

Van Hasselt, H., & Sutton, R. S. (2015). Learning to Predict Independent of Span. *ArXiv:1508.04582*.

Van Hasselt, H., Guez, A., Hessel, M., Mnih, V., & Silver, D. (2016). Learning Values Across Many Orders of Magnitude. *Advances in Neural Information Processing Systems*.

Van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., & Sutton, R. S. (2016). True Online Temporal-Difference Learning. *Journal of Machine Learning Research*.

Wainwright, M. J. (2019). Stochastic Approximation With Cone-Contractive Operators: Sharp ℓ_∞-Bounds for Q-learning. *ArXiv:1905.06265*.

Wan, Y., Naik, A., & Sutton, R. S. (2021a). Learning and Planning in Average-Reward Markov Decision Processes. *International Conference on Machine Learning*.

Wan, Y., Naik, A., & Sutton, R. S. (2021b). Average-Reward Learning and Planning with Options. *Advances in Neural Information Processing Systems*.

Wan, Y. (2023). *Learning and Planning with the Average-Reward Formulation*. Ph.D. dissertation, University of Alberta.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*.

Wheeler, R., & Narendra, K. (1986). Decentralized Learning in Finite Markov Chains. *IEEE Transactions on Automatic Control*.

White, A., & White, M. (2016). Investigating Practical Linear Temporal Difference Learning. *ArXiv:1602.08771*.

White, D. J. (1963). Dynamic Programming, Markov Chains, and the Method of Successive Approximations. *Journal of Mathematical Analysis and Applications*.

Wurman, P. R., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T. J., Capobianco, R., Devlic, A., Eckert, F., Fuchs, F., Gilpin, L., Khandelwal, P., Kompella, V., Lin, H., MacAlpine, P., Oller, D., Seno, T., Sherstan, C., Thomure, M. D., Aghabozorgi, H., Barrett, L., Douglas, R., Whitehead, D., Dürr, P., Stone, P., Spranger, M., & Kitano, H. (2022). Outracing Champion Gran Turismo Drivers With Deep Reinforcement Learning. *Nature*.

Xu, Z., van Hasselt, H., & Silver, D. (2018). Meta-Gradient Reinforcement Learning. *Advances in Neural Information Processing Systems*.

Yang, S., Gao, Y., An, B., Wang, H., & Chen, X. (2016). Efficient Average Reward Reinforcement Learning using Constant Shifting Values. *AAAI Conference on Artificial Intelligence*.

Young, K., & Tian, T. (2019). MinAtar: An Atari-Inspired Testbed for Thorough and Reproducible Reinforcement Learning Experiments. *ArXiv:1903.03176*.

Yu, H., & Bertsekas, D. P. (2009). Convergence Results for Some Temporal Difference Methods Based on Least Squares. *IEEE Transactions on Automatic Control*.

Yu, H. (2012). Least Squares Temporal Difference Methods: An Analysis under General Conditions. *SIAM Journal on Control and Optimization*.

Yu, H. (2017). On Convergence of Some Gradient-based Temporal-Differences Algorithms for Off-Policy Learning. *ArXiv:1712.09652*.

Zahavy, T., Xu, Z., Veeriah, V., Hessel, M., Oh, J., van Hasselt, H., Silver, D., & Singh, S. (2020). A Self-Tuning Actor-Critic Algorithm. *Advances in Neural Information Processing Systems*.

Zhang, R., Dai, B., Li, L., & Schuurmans, D. (2020a). GenDICE: Generalized Offline Estimation of Stationary Values. *ArXiv:2002.09072.*

Zhang, S., Liu, B., & Whiteson, S. (2020b). GradientDICE: Rethinking Generalized Offline Estimation of Stationary Values. *International Conference on Machine Learning*.

Zhang, S., Wan, Y., Sutton, R. S., & Whiteson, S. (2021). Average-Reward Off-Policy Policy Evaluation with Function Approximation. *International Conference on Machine Learning*.

Zhang, Y., & Ross, K. W. (2021). On-Policy Deep Reinforcement Learning for the Average-Reward Criterion. *International Conference on Machine Learning*.

Zhao, R., Abbas, Z., Szepesvári, D., Naik, A., Holland, Z., Tanner, B., & White, A. (2022). CSuite: Continuing Environments for Reinforcement Learning, *Github:* google-deepmind/csuite

Appendix A: Pseudocode

In this appendix, we provide the pseudocode of the average-reward algorithms proposed in Chapters 3 and 4. We also provide the pseudocode of discounted algorithms that use our proposed reward-centering idea from Chapter 5.

Algorithm 1: Differential TD-learning (one-step tabular off-policy prediction)

Input: The target policy π, the behavior policy b
Algorithm parameters: step-size parameters α, η
Initialize $V(s)\ \forall s,\ \bar{R}$ arbitrarily (e.g., to zero)
for *each time step* **do**
  Take action A according to b, observe R, S'
  $\rho = \pi(A|S)/b(A|S)$
  $\delta = R - \bar{R} + V(S') - V(S)$
  $V(S) = V(S) + \alpha\,\rho\,\delta$
  $\bar{R} = \bar{R} + \eta\,\alpha\,\rho\,\delta$
  $S = S'$
end

Algorithm 2: Differential Q-learning (one-step tabular off-policy control)

Input: The behavior policy b (e.g., ϵ-greedy)
Algorithm parameters: step-size parameters α, η
Initialize $Q(s,a)\ \forall s, a; \bar{R}$ arbitrarily (e.g., to zero)
Obtain initial S
for *each time step* **do**
  Take action A according to b, observe R, S'
  $\delta = R - \bar{R} + \max_a Q(S', a) - Q(S, A)$
  $Q(S, A) = Q(S, A) + \alpha\,\delta$
  $\bar{R} = \bar{R} + \eta\,\alpha\,\delta$
  $S = S'$
end

Algorithm 3: Linear Differential Q-learning

Input: The behavior policy b (e.g., ϵ-greedy)

Algorithm parameters: step-size parameters α, η

Initialize $\mathbf{w}_a \in \mathbb{R}^d \, \forall a, \bar{R}$ arbitrarily (e.g., to zero)

Obtain initial observation $\mathbf{x}$

for *all time steps* **do**

  Take action A according to b, observe $R, \mathbf{x}'$

  $\delta = R - \bar{R} + \max_a \mathbf{w}_a^\top \mathbf{x}' - \mathbf{w}_A \mathbf{x}$

  $\mathbf{w}_A = \mathbf{w}_A + \alpha \, \delta \, \mathbf{x}$

  $\bar{R} = \bar{R} + \eta \, \alpha \, \delta$

  $\mathbf{x} = \mathbf{x}'$

end

Algorithm 4: On-policy Linear Differential TD(λ) ("Algorithm 1")

Input: The target policy π

Algorithm parameters: step-size parameters α, η; trace parameter $\lambda \in [0, 1)$

Initialize $\mathbf{w} \in \mathbb{R}^d$, $\mathbf{z} \in \mathbb{R}^d$, $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., to zero)

Obtain initial state features $\mathbf{x}_S$

for *each time step* **do**

  Take action A according to π, obtain $R, \mathbf{x}_{S'}$

  $\delta = R - \bar{R} + \mathbf{w}^\top \mathbf{x}_{S'} - \mathbf{w}^\top \mathbf{x}_S$

  $\mathbf{z} = \lambda \mathbf{z} + \mathbf{x}_S$

  $\mathbf{w} = \mathbf{w} + \alpha \, \delta \, \mathbf{z}$

  $\bar{R} = \bar{R} + \eta \, \alpha \, \delta$

  $\mathbf{x}_S = \mathbf{x}_{S'}$

end

Algorithm 5: On-policy Linear Differential Sarsa(λ)

Input: The behavior policy π (e.g., ϵ-greedy)

Algorithm parameters: step-size parameters α, η; trace parameter $\lambda \in [0, 1)$

Initialize $\mathbf{w}_a \in \mathbb{R}^d \, \forall a$, $\mathbf{z} \in \mathbb{R}^d$, $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., to zero)

Obtain initial state features $\mathbf{x}_S$, select first action A according to π

for *each time step* **do**

  Take action A, obtain $R, \mathbf{x}_{S'}$, compute action A' according to π

  $\delta = R - \bar{R} + \mathbf{w}_{A'}^\top \mathbf{x}_{S'} - \mathbf{w}_A^\top \mathbf{x}_S$

  $\mathbf{z} = \lambda \mathbf{z} + \mathbf{x}_S$

  $\mathbf{w}_A = \mathbf{w}_A + \alpha \, \delta \, \mathbf{z}$

  $\bar{R} = \bar{R} + \eta \, \alpha \, \delta$

  $\mathbf{x}_S = \mathbf{x}_{S'}$ and $A = A'$

end

Algorithm 6: Off-policy Tabular Differential TD(λ) ("Algorithm 2")

Input: The target policy π, the behavior policy b

Algorithm parameters: step-size parameters α, η; trace parameter $\lambda \in [0, 1)$

Initialize $\mathbf{w} \in \mathbb{R}^{|\mathcal{S}|}$, $\mathbf{z} \in \mathbb{R}^{|\mathcal{S}|}$, $\bar{R} \in \mathbb{R}$, $z^{\bar{R}} \in \mathbb{R}$ arbitrarily (e.g., to zero)
Obtain initial state features $\mathbf{x}_S$
for *each time step* **do**
    Take action A according to b, obtain $R, \mathbf{x}_{S'}$
    $\delta = R - \bar{R} + \mathbf{w}^\top \mathbf{x}_{S'} - \mathbf{w}^\top \mathbf{x}_S$
    $\rho = \pi(A|S)/b(A|S)$
    $\mathbf{z} = \rho(\lambda \mathbf{z} + \mathbf{x}_S)$
    $z^{\bar{R}} = \rho(\lambda z^{\bar{R}} + 1)$
    $\mathbf{w} = \mathbf{w} + \alpha\, \delta\, \mathbf{z}$
    $\bar{R} = \bar{R} + \eta\, \alpha\, \delta\, z^{\bar{R}}$
    $\mathbf{x}_S = \mathbf{x}_{S'}$
end

Algorithm 7: Tabular Q-learning with value-based reward centering

Input: The behavior policy b (e.g., ϵ-greedy)

Algorithm parameters: discount factor γ, step-size parameters α, η

Initialize $Q(s, a)\ \forall s, a$; $\bar{R}$ arbitrarily (e.g., to zero)
Obtain initial S
for *all time steps* **do**
    Take action A according to b, observe R, S'
    $\delta = R - \bar{R} + \gamma \max_a Q(S', a) - Q(S, A)$
    $Q(S, A) = Q(S, A) + \alpha\, \delta$
    $\bar{R} = \bar{R} + \eta\, \alpha\, \delta$
    $S = S'$
end

Algorithm 8: Linear Q-learning with value-based reward centering

Input: The behavior policy b (e.g., ϵ-greedy)

Algorithm parameters: discount factor γ, step-size parameters α, η

Initialize $\mathbf{w}_a \in \mathbb{R}^d\ \forall a$, $\bar{R}$ arbitrarily (e.g., to zero)
Obtain initial observation $\mathbf{x}$
for *all time steps* **do**
    Take action A according to b, observe $R, \mathbf{x}'$
    $\delta = R - \bar{R} + \gamma \max_a \mathbf{w}_a^\top \mathbf{x}' - \mathbf{w}_A \mathbf{x}$
    $\mathbf{w}_A = \mathbf{w}_A + \alpha\, \delta\, \mathbf{x}$
    $\bar{R} = \bar{R} + \eta\, \alpha\, \delta$
    $\mathbf{x} = \mathbf{x}'$
end

Algorithm 9: (Non-linear) DQN with value-based reward centering

Input: The behavior policy b (e.g., ϵ-greedy)

Algorithm parameters: discount factor γ, step-size parameters α, η

Initialize value network, target network; initialize $\bar{R}$ arbitrarily (e.g., to zero)
Obtain initial observation $\mathbf{x}$
for *all time steps* **do**
    Take action A according to b, observe $R, \mathbf{x}'$
    Store tuple $(\mathbf{x}, A, R, \mathbf{x}')$ in the experience buffer
    if *time to update estimates* **then**
        Sample a minibatch of transitions $(\mathbf{x}, A, R, \mathbf{x}')^b$
        For every i-th transition: $\delta_i = R_i - \bar{R} + \gamma \max_a \hat{q}(\mathbf{x}'_i, a) - \hat{q}(\mathbf{x}_i, A_i)$
        Perform a semi-gradient update of the value network parameters with a loss function of δ^2
        $\bar{R} = \bar{R} + \eta\, \alpha\, \text{mean}(\delta)$
        Update the target network occasionally
    end
    $\mathbf{x} = \mathbf{x}'$
end

Appendix B: Additional Experimental Details

In this appendix, we provide the additional results and experimental details for Chapter 5.

Table B.1 contains a list of all the hyperparameters tested for the hyperparameters that are common across all the domains: γ, α, η. Note that if we initialize the reward-rate estimate to zero and set $\eta = 0$ when using Q-learning with reward centering, we get the standard uncentered Q-learning algorithm. Also note that DQN with centering (in its current form) requires a larger value of η compared to the the tabular or linear versions because of how a minibatch is used in the implementation of this deep RL algorithm. In line 11 of Algorithm 9, the mean of the TD errors of the minibatch of transitions is taken. The mean can make the overall gradient for the reward-rate update very small, so a large value of η can be used.

The number of timesteps, number of runs, initializations are reported in the main text. The agent's behavior policy was always ϵ-greedy with fixed $\epsilon = 0.1$. We set commonly used values for the various parameters of the deep RL (non-linear) experiments: the batch size was 64, the value-network and reward-rate parameters were updated every 32 steps, the target network was updated every 128 steps, the experience buffer size was 10,000. Apart for the main step-size parameter, the default parameters (set by PyTorch (Paszke et al., 2019)) were used for the Adam optimizer.

Table B.1: List of hyperparameters tested for each domain

	γ	α	η
Access-Control Queuing (tabular)	[0.5, 0.8, 0.9, 0.99, 0.999, 1]	[1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2, 1]	[0, 1/256, 1/64, 1/16, 1/4, 1]
PuckWorld (linear)	[0.5, 0.8, 0.9, 0.99, 0.999, 1]	[0.01, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.1]	[0, 1/256, 1/64, 1/16, 1/4, 1]
Catch (linear)	[0.5, 0.8, 0.9, 0.99, 0.999, 1]	[1/128, 1/64, 1/32, 1/16, 1/8, 1/4, 1/2, 1]	[0, 1/256, 1/64, 1/16, 1/4, 1]
Catch (non-linear)	[0.5, 0.8, 0.9, 0.99, 0.999, 1]	[1/512, 1/256, 1/128, 1/64, 1/32, 1/16, 1/8]	[0, 1, 2, 4, 8, 16]
Pendulum (non-linear)	[0.5, 0.8, 0.9, 0.99, 0.999, 1]	[1/512, 1/256, 1/128, 1/64, 1/32, 1/16, 1/8]	[0, 1, 2, 4, 8, 16]

In our implementations we added two simple optimizations:

1. Make the reward-rate estimate completely independent of its initialization: this can be done using the unbiased constant step-size trick (see Sutton & Barto's (2018) Exercise 2.7).

2. Propagate the changes to the reward-rate estimate faster: this can be done by first computing the TD error, then updating the reward-rate estimate, then recomputing the TD error with the new reward-rate estimate, and finally updating the value estimate(s).

These optimizations did not affect the overall trends in the results but provided a small yet noticeable improvement for a tiny computational cost, hence we recommend using them.

For the experiments involving a shift in the problem rewards, the rewards obtained on each problem variant are not directly comparable. For intuition, imagine the first four rewards in the original problem be 2,0,3,1. In a variant of the problem with 5 added to all the rewards, the first four rewards may now appear to be 7,5,8,4. An agent solving the latter problem might trivially appear better than one solving the

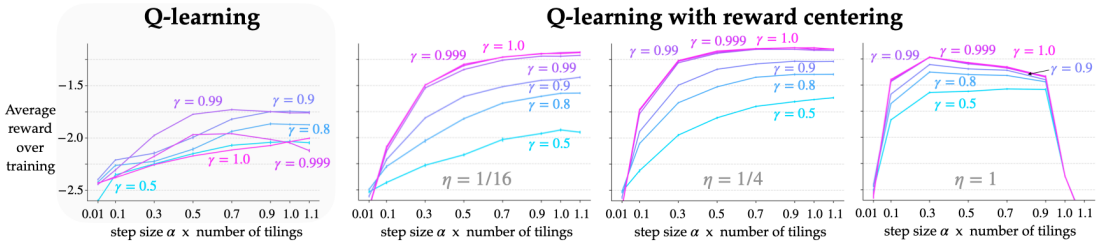


Figure B.1: Parameter studies showing the performance sensitivity of the algorithms to their parameters on the PuckWorld problem. *Far left:* Q-learning's performance was relatively poor for a large range of α. *Center to right:* For each discount factor, the performance of Q-learning with centering was better across a broad range of α. Moreover, performance only changed a little w.r.t. η.

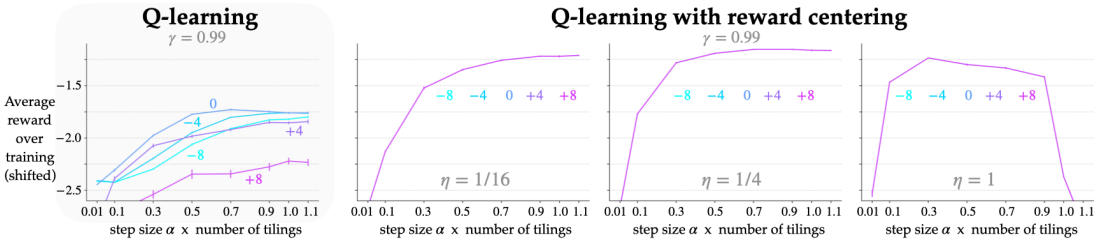


Figure B.2: Parameter studies showing the performance sensitivity of the algorithms to variants of the PuckWorld problem. The error bars indicate one standard error, which at times is less than the width of the lines. *Far left:* Q-learning's performance differed significantly on the different variants over a broad range of the step-size parameter α. *Center to right:* With centering, the performance was about the same across the problem variants, and was quite robust to the choice of its parameter η. All the curves correspond to $\gamma = 0.99$; the trends were consistent across other discount factors.

former problem even though its fourth reward was relatively lower. To compare them meaningfully, from the rewards obtained by an agent, we can subtract the constant that was added in the first place to all the problem's rewards. That is, we can shift the rewards back to make fair comparisons across problem variants. This is what we did when presenting the results of the shifting experiments; this is explicitly denoted by the word "shifted" in the y-axis label.

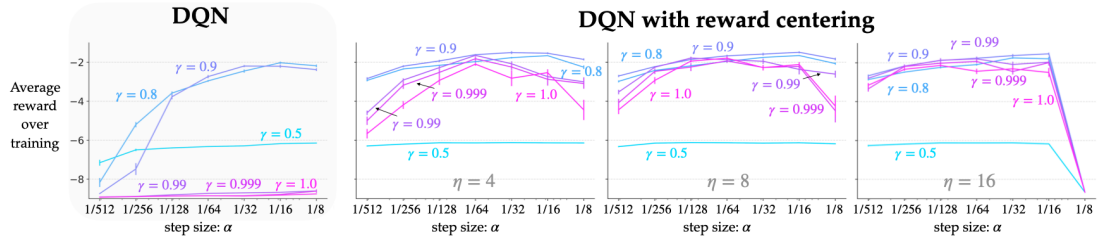


Figure B.3: Parameter studies showing the performance sensitivity of the two algorithms to their parameters on the Pendulum problem. *Far left:* The performance of DQN was not good for discount factors larger than 0.9. *Center to right:* For each discount factor, the performance of Q-learning with centering was better across a broad range of α, and the performance was not too sensitive to its second parameter η. $\gamma = 0.5$ was too small to solve this problem.

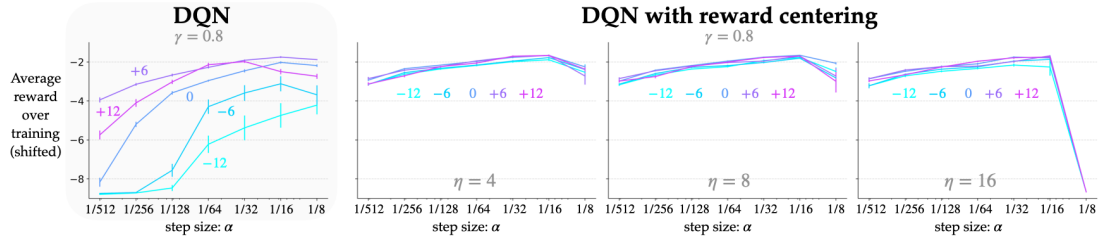


Figure B.4: Parameter studies showing the performance sensitivity of the two algorithms with $\gamma = 0.8$ to variants of the Pendulum problem. *Far left:* DQN's performance differed significantly on the different variants. *Center to right:* The performance of DQN with centering was about the same across the problem variants across a large range of the step size α, and was also quite robust to the choice of η.