University of Alberta

Gradient Temporal-Difference Learning Algorithms

by

Hamid Reza Maei

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Examining Committee

Abstract

We present a new family of gradient temporal-difference (TD) learning methods with function approximation whose complexity, both in terms of memory and per-time-step computation, scales linearly with the number of learning parameters. TD methods are powerful prediction techniques, and with function approximation form a core part of modern reinforcement learning (RL). However, the most popular TD methods, such as TD(λ), Q-learning and Sarsa, may become unstable and diverge when combined with function approximation. In particular, convergence cannot be guaranteed for these methods when they are used with off-policy training. Off-policy training—training on data from one policy in order to learn the value of another—is useful in dealing with the exploration-exploitation tradeoff. As function approximation is needed for large-scale applications, this stability problem is a key impediment to extending TD methods to real-world large-scale problems. The new family of TD algorithms, also called *gradient-TD* methods, are based on stochastic gradient-descent in a Bellman error objective function. We provide convergence proofs for general settings, including off-policy learning with unrestricted features, and nonlinear function approximation. Gradient-TD algorithms are on-line, incremental, and extend conventional TD methods to off-policy learning while retaining a convergence guarantee and only doubling computational requirements. Our empirical results suggest that many members of the gradient-TD algorithms may be slower than conventional TD on the subset of training cases in which conventional TD methods are sound. Our latest gradient-TD algorithms are "hybrid" in that they become equivalent to conventional TD—in terms of asymptotic rate of convergence—in on-policy problems.

Acknowledgements

This PhD thesis is developed through collaboration with several great scientists. First, I would like to thank Richard S. Sutton and Csaba Szepesvári, who have made significant contributions in this work. Rich is a deep thinker and is one of the founders of reinforcement learning (RL). Rich was not only my PhD supervisor, but he was also a mentor. He has been very kind and supportive during all these years. We spent enormous amounts of time together to build a robust temporal-difference (TD) learning algorithm that is scalable to large-scale applications and is convergent under off-policy setting. We tried a lot of different ideas; many of them failed. Through this trial and error procedure, I was able to grasp a lot of deep concepts in RL, particularly in TD learning. And, finally we developed gradient TD algorithms. Csaba also helped us a lot and played a major role in improving our theoretical results. Csaba is a world-class scientist and is one of the pioneers in theoretical RL. He was extremely kind, and helped me a lot in every aspect that I can think of. We had a lot of interesting meetings together. Most of the times, first, we made an espresso, which made me to follow the math easier! I feel very lucky to have him as one of my major collaborators.

I wish to express my gratitude to Doina Precup and Shalabh Bhatnagar, who also contributed to this work in various ways and provided some great insights. I wish to thank my examining committee for reading the manuscript and their comments. I would like also to thank Dale Schuurmans and Geoffrey J. Gordon. Dale and Geoff gave me detailed and insightful comments on the manuscript. I would like to thank the RLAI lab members. Particularly, I wish to thank Eric Wiewiora, Istvan Szita, Ashique Mahmood (Rupam), Patrick Pilarski, Joel Veness, and David Silver. Patrick and Rupam also provided some detailed comments on this manuscript. Thanks to David Silver for providing Computer Go results on gradient-TD methods.

Finally, I would like to express my special loving thanks to my wife, Farzaneh Foroodi Nejad, whose love and support was always with me during my graduate studies. She was one of the main reasons that I chose to come to University of Alberta and join to RLAI group, which has been a great and fruitful experience. Without her kind support and caring, I was not able to complete this work on time. My deepest thanks also go to my parents for their love and support.

Financial support for this work, mainly, has been provided by Alberta Innovates Graduate Student Scholarship (formerly iCore).

Contents

List of Figures

Chapter 1

Introduction

Temporal-difference (TD) methods are central and powerful techniques in solving reinforcement learning (RL) problems (Sutton and Barto, 1998). TD learning—in contrast to Monte-Carlo— has the ability to learn from incomplete sequence of events without waiting for the final outcome. This key property allows moment-by-moment (real-time) predictions. For example, consider playing chess. In order to learn and make predictions, the player does not need to wait until the game ends. Instead, the player updates her predictions about the outcome after each move. TD learning is a bootstrapping method; that is, the current prediction is updated based on the next prediction. If the difference between the two predictions are positive, then it increases the prediction of winning for situations similar to the current situation.

In many problems of interest, however, the number of situations—*states*— is very large. For example, in games such as Backgammon, Chess, and 19×19 computer Go, there are roughly 10^{28}, 10^{47} and 10^{170} states, respectively. Clearly, it is not practical to compute the value of each state individually (this problem is also related to the curse-of-dimensionality). To solve this problem we need to abandon the idea of computing value functions exactly and instead use approximation techniques. This procedure is known as function approximation.

Function approximation, as a generalization technique, allows us to approximate the value of unseen states from observed data. More specifically, the states are mapped to feature vectors that have many fewer components than the number of states. To estimate the approximate value of each state, function approximation combines (e.g. linearly) the state's feature vector with a parameter vector of the same size.

However, many popular TD algorithms, including TD(λ), Q-learning and Sarsa, when combined with function approximation, can become unstable and diverge—their learning parameters will go to infinity (Baird, 1995; Tsitsiklis and Van Roy, 1997). The only convergence guarantee that has been reported for TD(λ) with *linear* function approximation is restricted to the case where states are sampled according to the policy being evaluated;

that is, (state) *trajectory following*. This training scenario is called on-policy learning (Sutton 1988; Tsitsiklis and Van Roy, 97; Tadić, 2001). Nevertheless, there exists a number of counterexamples for which TD(λ) with linear function approximation may diverge under off-policy training (Baird, 1995; Sutton and Barto, 1998).

Off-policy learning refers to learning about one way of behaving, called the *target* policy, from data generated by another way of selecting actions, called the *behavior* policy. The target policy is often a deterministic policy that approximates the optimal policy. Conversely, the behavior policy is often stochastic, exploring all possible actions in each state as part of finding the optimal policy. Freeing the behavior policy from the target policy allows a greater variety of exploration strategies to be used. It also enables learning from training data generated by unrelated controllers, including manual human control, and from previously collected data. A third reason for interest in off-policy learning is that it permits learning about multiple target policies (e.g., optimal policies for multiple sub-goals) from a single stream of data generated by a single behavior policy, in other words, parallel learning.

The stability problem of popular TD methods with function approximation indicates that there is a severe problem with the way table-lookup TD methods are combined with function approximation. Particularly, this stability problem arises when we seek the following four desirable algorithmic features: (1) TD learning, (2) function approximation, (3) off-policy learning, (4) linear complexity both in terms of memory and per-time-step computation. Finding a temporal-difference algorithm that is effective on large applications with function approximation has been one of the important open problems in reinforcement learning for more than a decade.

The stability problem is not specific to reinforcement learning. Classical dynamic programming methods such as value and policy iteration are also off-policy methods and also may diverge when they are used in conjunction with linear or non-linear function approximation.

1.1 Related works

Several approaches to the stability problem in reinforcement learning have been proposed, but none have been satisfactory in many ways, such as high variance, inferior solution, or high complexity in terms of memory and per-time-step computation.

One idea for retaining all four desirable features is to use cumulative products of target-to-behavior-policy likelihood ratios to re-weight TD updates. In principle it uses importance sampling idea over the whole data trajectory (not just on a given sample). In other words, at any given time t, the TD update is re-weighted by cumulative products of importance sampling ratios (likelihood ratios) up to time t, so that its expected value is in the same direction as on-policy update according to the target policy (Precup et al., 2000; 2001). Convergence can sometimes then be assured by existing results on the convergence of on-policy methods (Tsitsiklis and Van Roy 97; Tadić 2001). However, the importance sampling weights are cumulative products of many target-to-behavior-policy likelihood ratios, and consequently they, and the corresponding updates, may be of very high variance.

The use of "recognizers" to construct the target policy directly from the behavior policy (Precup et al., 2006) is one strategy for limiting the variance; another is careful choice of the target policies (Precup et al., 2001). However, it remains the case that for all of such methods to date there are always choices of problem, behavior policy, and target policy for which the variance is infinite, and thus for which there is no guarantee of convergence.

The residual gradient (RG) method (Baird, 1995) has also been proposed as a way of obtaining all four desirable features. The RG algorithm has a convergence guarantee, however, it has two major problems.

The first problem with RG method is that it requires two independent samples for the next state given the current state. This is known as double-sampling. If the model of the environment is known, then two independent samples can be generated from the model. To overcome the double-sampling problem—for the case where the model of the environment is not known or computationally is expensive to learn— Baird (1995) recommended using only one sample for the next state. This method can be viewed as gradient-descent in the expected squared TD-error. It converges stably to a solution, that minimizes mean-squared TD-error given an arbitrary differentiable function approximation. However, this solution has always been found to be inferior to the TD solution (see Sec. 3.2).

The second problem that seems to be fundamental is that, for the case of function approximation, sometimes the solution obtained from RG method is not reliable even if the algorithm has access to two independent samples. I will discuss about this in Sec. 3.2.

Gordon (1995) and others have questioned the need for linear function approximation. He has proposed replacing linear function approximation with a more restricted class of approximation, known as *averagers*, that never extrapolate outside the range of the observed data and thus cannot diverge (see also Szepesvari & Smart, 2004). Rightly or wrongly, averagers have been seen as being too constraining and have not been used on large applications involving online learning. Linear methods, on the other hand, have been widely used (e.g. Baxter, Tridgell & Weaver, 1998; Schaeffer, Hlynka & Jussila, 2001).

The need for linear complexity has also been questioned. Second-order methods for linear approximations, such as LSTD (Bradtke and Barto, 1996; Boyan, 2002) and LSPI (Lagoudakis and Parr, 2003), can be effective on moderately sized problems. If the number of features in the linear approximation is d, then these methods require memory and per-time-step computation in the order d^2. Newer incremental methods such as iLSTD (Geramifard et al., 2006) have reduced the per-time-complexity to $O(kd)$, where k is a moderately chosen natural number, but iLSTD still has complexity of $O(d^2)$ in memory. Sparsification methods may reduce the complexity further, they do not help in the general case, and may apply to $O(d)$ methods as well to further reduce their complexity.

Linear function approximation is most powerful when very large numbers of features are used, perhaps millions of features (e.g., as in Silver et al., 2007). In such cases, methods with $O(d)$ complexity are required.

The stability problem of TD methods in conjunction with nonlinear function approximation is more severe than with linear function approximation, as conventional TD methods with nonlinear function approximation may diverge even for the case of on-policy learning (Tsitsiklis and Van Roy, 1997). There has been little work on addressing this stability problem, and proposed methods either are restricted to particular conditions or only partially solve the problem. For example, the Bridge algorithm by Papavassiliou and Russell (1999) uses a variant of TD learning and is convergent with nonlinear function approximation. However, it is a complex algorithm with high computational complexity, which hampers its practicality and also it does not have all of our four desirable algorithmic features.

1.2 Contributions

In this thesis, I present a new family of TD algorithms based on gradient-descent. These algorithms are the first TD methods that have all of the four desirable algorithms features while retaining a convergence guarantee.

Our new algorithms are based on gradient-descent in a Bellman-error objective function and as a result we call them *gradient-TD* methods. Gradient-TD algorithms can be seen as an alternative conventional TD methods, such as TD(λ), Sarsa and Q-learning (Watkins, 1989). A distinctive feature that distinguishes gradient-TD methods from conventional TD algorithms is the use of a second set of weights (auxiliary weights) in their main update rule.

In addition, we make the following extensions:

- We extend our gradient-TD algorithms to include eligibility traces. Particularly, we let the eligibility trace parameter, λ, be an arbitrary function of state.

- We extend our gradient-TD algorithms to include general policy termination probability. This extension makes our algorithms suitable for learning general value functions (GVFs).

The result of the above extensions is introducing two novel gradient-TD algorithms: GTD(λ) and GQ(λ). These extensions bring us closer to the ultimate goal of this work—the development of a general prediction learning algorithm suitable for learning experientially grounded knowledge of the world.

Although GQ(λ) is developed for the prediction domain, it can be extended to the control domain. Thus, we introduce Greedy-GQ(λ), and also Greedy-GQ for the case of λ=0, in which the target policy is greedy with respect to the current estimate of the value function. Greedy-GQ can be seen as Q-learning with a gradient correction term. In this thesis, however, we do not present convergence proof for Greedy-GQ(λ), because the proof is not in a ready state. However, to get a flavour of the convergence proof, we refer the reader to Maei et al. (2010).

Some of our empirical results suggest that gradient-TD method maybe slower than conventional TD methods on problems on which conventional TD methods are sound (that is, on-policy learning problems).

Our latest gradient-TD algorithms, called *hybrid* gradient-TD methods, potentially can improve the rate of convergence. They are "hybrid" in that they become equivalent to (and thus just as fast as) conventional TD in on-policy cases in terms of asymptotic rate of convergence. We briefly mention these hybrid algorithms in Appendix C.

Figure 1.1 gives a general picture of the problem of TD methods with function approximation, and Figure 1.2 summarizes the new gradient-TD algorithms that we present in this thesis. The figures show that TD methods are robust only with exact tabular representations. When combined with function approximation they may diverge, and their convergence can be guaranteed only for a narrow class of problems, such as on-policy learning, along with a narrow class of function approximations, such as linear function approximation.

All of our algorithms are based on gradient descent, and are provably convergent.

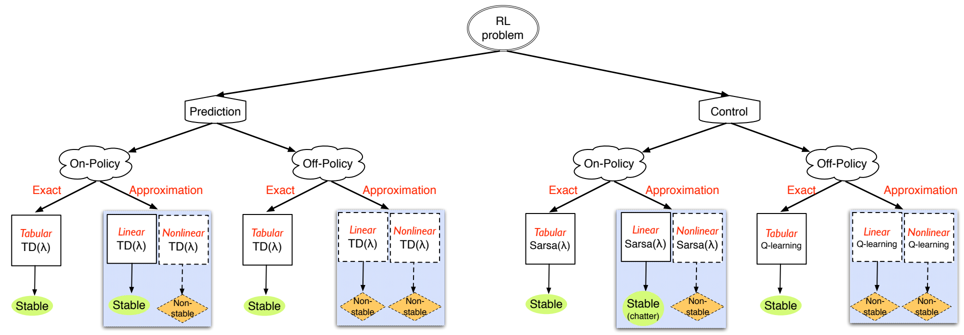

Figure 1.1: Status of conventional TD methods with tabular representation and function approximation, in terms of stability. For stability analysis of linear Sarsa (λ) as well as its chattering behavior see Gordon (2000).

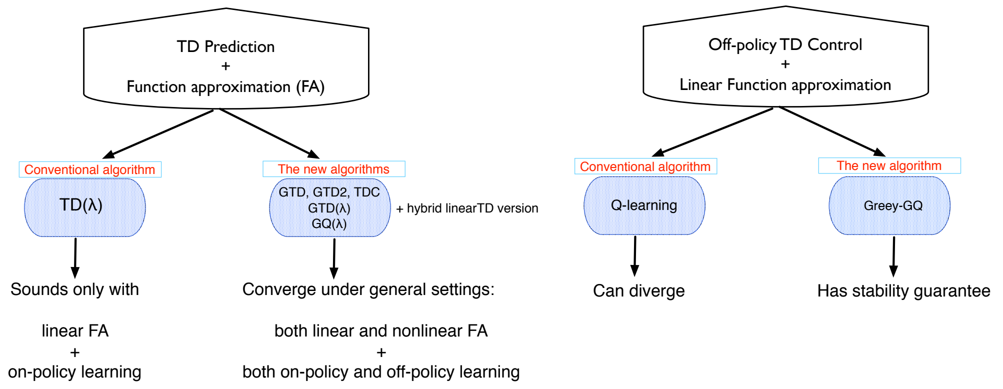

Figure 1.2: Algorithmic contributions.

Chapter 2

Background

TD learning is central in reinforcement learning due to its bootstrapping and prediction ability. As such, TD learning has been used for prediction problems, such as policy evaluation, in reinforcement learning. In addition, TD methods have been extended to problems that involve finding a good policy to achieve a goal (Sutton and Barto, 1998; Szepesvári, 2010).

In this chapter, I give an overview of the most popular and classical temporal-difference (TD) learning methods, such as TD(λ), Q-learning and Sarsa. I will show that popular TD methods, which are based on gradient-descent, are not true gradient-descent methods, and as such the condition under which they converge is very narrow.

2.1 The reinforcement learning problem

The problem of reinforcement learning (RL) refers to the problem of learning by interaction with an environment to accomplish a goal. The *environment* is everything that learner interacts with. The learner, also called *agent*, makes a decision by taking an action at every time-step according to a *policy*, and the environment emits a scalar reward for every action. The ultimate goal of RL agent is to take sequence of actions that maximize the received rewards in the long run.

In the standard RL framework there are two distinct problems: 1) *Policy evaluation* or *prediction*, which refers to evaluating the consequences of following a fixed policy, and 2) *Control*, which refers to finding an optimal (or near optimal) policy that maximizes the expected future reward. For some of the practical applications of RL methods, we refer the reader to Tesauro (1992), Crites & Barto (1995), Marbach et al. (1997), and Peters et al. (2005).

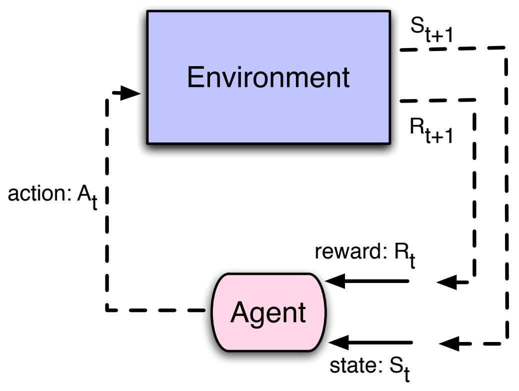

Figure 2.1: The agent and environment interaction in conventional reinforcement learning setting. The t index indicates time-step.

2.2 The problem formulation

To formulate the reinforcement learning problem, we consider the standard RL framework in which a learning agent interacts with an environment consisting of a finite Markov decision process (MDP). At each of a sequence of discrete time-steps, $t = 0, 1, 2, \ldots$, the environment is in a state $S_t \in \mathcal{S}$, the agent chooses an action $A_t \in \mathcal{A}$, and then the environment emits a reward $R_{t+1} \in \mathbb{R}$, after which the agent find itself in a next state $S_{t+1} \in \mathcal{S}$ (see Figure 2.1). The state and action sets are finite. State transitions are stochastic and dependent on the immediately preceding state and action and rewards are stochastic and dependent on the preceding state and action, and on the next state.

We mentioned that the goal of RL agent is to maximize received rewards in the long run, but how can we formulate long run reward? A standard approach is the value function based approach. The state-value function of a policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, is defined as:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | S_0 = s, \pi\right]. \tag{2.1}$$

where $\gamma \in [0, 1)$ is the *discount rate* and $\mathbb{E}[.]$ denotes expectation over random samples, which are generated by following policy π. In this thesis, random variables (e.g. state, action and reward) are denoted by capital letters.

Let P^{π} denote state-state transition probability matrix and $V^{\pi} \in \mathbb{R}^{|\mathcal{S}|}$ be value function vector, whose s^{th} element is $V^{\pi}(s)$. It is well known that $V^{\pi} \in \mathbb{R}^{|\mathcal{S}|}$ satisfies the following

Bellman equation:

$$\begin{aligned} V^\pi &= R^\pi + \gamma P^\pi V^\pi \\ &\overset{\text{def}}{=} T^\pi V^\pi, \end{aligned} \tag{2.2}$$

where R^π is the vector with components $\mathbb{E}[R_{t+1}|S_t = s]$, and T^π is known as the *Bellman operator*.

Analogously, we can define the action-values, $Q^\pi(s, a)$, which evaluates the value of taking action a from state s:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}|S_0 = s, A_0 = a, \pi\right]. \tag{2.3}$$

So far we have formulated the problem of policy evaluation or prediction. However, solving a reinforcement learning problem roughly means finding an *optimal policy* that achieves lots of rewards in the long run. An optimal policy is a policy that its value function is at least better or equal than the value function of other policies. The value function of an optimal policy is called the *optimal value function*. The optimal state-value function, denoted V^*, is defined as:

$$V^*(s) \overset{\text{def}}{=} \max_\pi V^\pi(s), \quad \forall s \in \mathcal{S},$$

and the *optimal action-value* function, denoted Q^* is

$$Q^*(s, a) \overset{\text{def}}{=} \max_\pi Q^\pi(s, a), \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}.$$

In the next section, we review some of the most popular temporal-difference (TD) learning algorithms, also known as classical TD methods. One of the key property of classical TD methods is their ability to learn from every single fragment of experience without waiting for the final outcome.

2.3 Temporal-difference learning

TD learning is a key idea for prediction and plays central role in reinforcement learning (Sutton, 1988; Sutton and Barto, 1998). It uses the bootstrapping ideas developed in dynamic programming as well as Monte Carlo ideas. Classical TD methods such as TD(λ), Sarsa, and Q-learning are simple, sample-based, online, and incremental algorithms and as such are popular in the RL community.

The next two subsections give an overview of TD learning methods for the prediction (policy evaluation) and control problems.

2.3.1 Temporal-difference learning for prediction

TD learning can be used for evaluating the value of a fixed policy, π. This is known as prediction or policy evaluation. TD methods use each fragment of experience to update the value of state S_t, $V_t(S_t)$, at time t. This would allow moment-to-moment prediction. This is different than dynamic programming (DP) approach in the sense that the value of each state, in DP approach, are updated by sweeping over next states. In the next paragraph, we present the simplest TD method.

The tabular TD(0) algorithm for estimating V^π: The simplest TD method, known as tabular *TD(0)*, estimates the value of each individual state; e.g. S_t, according to the following update:

$$\begin{aligned} V_{t+1}(S_t) &= V_t(S_t) + \alpha_t \left[R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t) \right], \\ &= V_t(S_t) + \alpha_t \delta_t, \end{aligned} \tag{2.4}$$

where

$$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t), \tag{2.5}$$

is one-step TD error, or in short *TD error*, and α_t is a deterministic positive step-size parameter, which is typically small, or for the purpose of convergence analysis is assumed to satisfy the Robbins-Monro conditions: $\sum_{t=0}^{\infty} \alpha_t = \infty$, $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$. Tabular TD(0) is guaranteed to converge to V^π under standard conditions.

2.3.2 Temporal-difference learning for control

TD learning also can be used for finding an optimal policy. The problem of finding an optimal policy is known as control problem. One way to find an optimal policy is through value function based approach. That is, we can find the optimal policy through its value. Two popular TD methods have been suggested for this problem (see Sutton and Barto, 1988):

Sarsa: On-policy TD control The update for Sarsa the algorithm is similar to TD(0) in the sense that instead of state values, Sarsa updates action-values (also called Q-functions) according to

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha_t \left[R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t) \right].$$

One natural choice for choosing policy π, which generates actions is ϵ-greedy policy due to exploration-exploitation trade-off. The theorems, which have been used for the convergence of TD(0), also can be used for the convergence of Sarsa.

Q-learning: Off-policy TD control One of the breakthrough algorithms in RL is Q-learning. Q-learning also is one of the most popular algorithms among TD methods because: (1) It is simple, (2) It can find optimal action-values under off-policy training. The action-value update is according to

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha_t \left[R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a) - Q_t(S_t, A_t) \right].$$

Q-learning is off-policy because the agent learns about a policy that differs from the policy it is following. That is, at time t the agent is in state S_t and takes action, A_t, according to the policy π, and the environment takes the agent to state S_{t+1}. The agent, however, updates its action-values by learning about the value of a greedy action, and not the value of executed actions.

The Q-learning algorithm that I described in this section is in its tabular form; that is, the action-value function is computed for each state-action pair. Tabular Q-learning is guaranteed to converge to optimal action-values if states are visited infinitely (Sutton & Barto, 1998).

2.4 TD learning with eligibility traces

Classical TD methods that we talked about in previous sections, only use one-step back-up to update the value function. However, it is possible to look further ahead. Eligibility traces, allow us to look further ahead through their mechanistic backward-view updates. Several important properties of eligibility traces are as follows:

- They bridge the temporal gaps in cause and effect when experience is processed at a temporally fine resolution.
- They make classical TD methods more like efficient incremental Monte-Carlo algorithms. For example, in TD(λ), $\lambda \in [0, 1]$ refers to eligibility function and is

equivalent to Monte-Carlo methods when $\lambda = 1$.

- They are particularly of interest when reward is delayed by many steps, thus, by adjusting λ function we may get faster and efficient learning.

Here, $e_t(s)$ is called eligibility trace for state s at time t, and for tabular representation is defined as

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq S_t; \\ 1 + \gamma\lambda e_{t-1}(s) & \text{if } s = S_t, \end{cases}$$

TD(λ), Sarsa(λ) and Q(λ) are classical TD algorithms with eligibility traces. For example, TD(λ) update is as follows:

$$V_{t+1}(s) = V_t(s) + \alpha_t \delta_t e_t(s), \quad \forall s \in \mathcal{S}. \tag{2.6}$$

Similar updates exist for Sarsa(λ) and Q(λ). For the detailed description of these algorithms we refer the reader to Sutton & Barto (1998).

Figure 2.2, illustrates various methods used for value function estimation. As we can see, TD learning has an interesting property, which allows it to learn from a single transition, while Monte-Carlo method needs to wait until it observes the outcome, and then it updates. Each method, has its own advantage, and thus, TD(λ) has been developed to bridge the gap between the two methods.

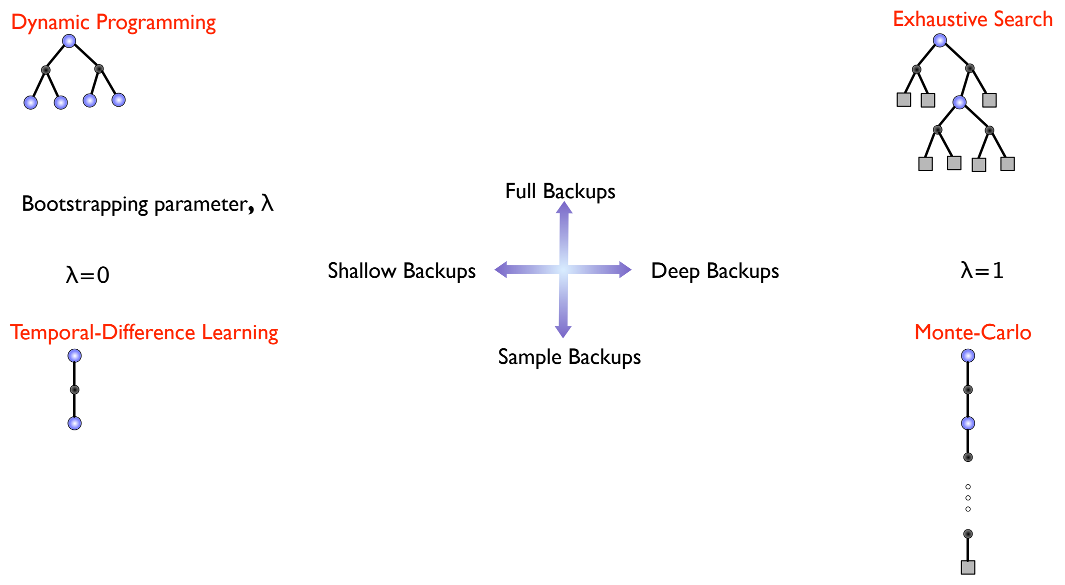

Figure 2.2: The space of methods. The small (black) blobs represent actions while the large blobs represent states. The squares represent the absorbing states (or terminal states).

2.5 Temporal-Difference learning with function approximation

The TD(0) algorithm (2.4), can be combined with parametrized value function V_θ, $\theta \in \mathbb{R}^d$. The value function can be either linear or nonlinear and differentiable function (such as neural networks) with respect to the parameter vector, θ. The resulting algorithm has the following update rule:

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t(\theta_t) \nabla V_{\theta_t}(S_t), \tag{2.7}$$

where

$$\delta_t(\theta_t) = R_{t+1} + \gamma V_{\theta_t}(S_{t+1}) - V_{\theta_t}(S_t), \tag{2.8}$$

and $\nabla V_\theta(s) \in \mathbb{R}^d$ denotes the gradient of V_θ with respect to θ at s. We call this algorithm linear/nonlinear TD(0). Also, the TD(0)-solution (or TD(0)-fixpoint), θ, (if it exists), satisfies:

$$\mathbb{E}[\delta_t(\theta) \nabla V_\theta(S_t)] = 0. \tag{2.9}$$

Note, for simplicity we adopt the following notation for TD error:

$$\delta_t \equiv \delta_t(\theta_t).$$

Sutton's TD(λ) with linear function approximation (Sutton, 1988) is one of the simplest form of TD learning with function approximation and, since its development, has played central role in modern reinforcement learning. To start, first we overview the simplest form of TD(λ) with linear function approximation, and we call it *linear TD(0)*. In the beginning, let's limit ourselves to on-policy training data. In Chapter 4, we show how to extend TD ideas to off-policy learning.

The linear TD(0) algorithm, starts with an arbitrary parameter vector, θ_0. Upon observing the t^{th} transition from state S_t to S_{t+1} (on-policy transitions), which follows with feature-vector observation $(\phi_t, R_{t+1}, \phi_{t+1})$, where $\phi_t \equiv \phi(S_t)$, the learning parameter vector is updated according to

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \phi_t, \tag{2.10}$$

where $\delta_t = R_{t+1} + \gamma \theta_t^\top \phi_{t+1} - \theta_t^\top \phi_t$ is TD-error with linear function approximation. If the discount factor γ is zero, the problem becomes supervised learning and linear TD(0) update rule becomes the conventional least-mean-square (LMS) rule in supervised learning. Thus, the key feature that distinguishes reinforcement learning from supervised learning is the existence of bootstrapping term, $\theta_t^\top \phi_{t+1}$, in the update rule (2.10). This will allow the algorithm to guess about future outcome without waiting for it—allows it to learn from single fragment of experience without waiting for the final outcome. Also, this is the key difference between TD learning and conventional Monte-Carlo methods (which are based on supervised learning ideas). Thus, TD methods have the ability to learn from single transitions without waiting for the final outcome; they do this by guessing from a guess! We will see how this key idea allows TD to learn from off-policy data.

It is well known that the linear TD(0) algorithm is convergent under on-policy training (Tsitsiklis and Van Roy,1997; Tadić, 2001). From the theory of stochastic methods, the convergence point of linear TD(0), is a parameter vector, say θ, that satisfies

$$0 = \mathbb{E}[\delta_t(\theta)\phi_t] = b - A\theta, \tag{2.11}$$

where

$$\delta_t(\theta) = R_{t+1} + \gamma\theta^\top\phi_{t+1} - \theta^\top\phi_t, \tag{2.12}$$

$$A = \mathbb{E}\Big[\phi_t(\phi_t - \gamma\phi_{t+1})^\top\Big], \quad b = \mathbb{E}[R_{t+1}\phi_t], \tag{2.13}$$

and the expectation is over all random samples. In this thesis, parameter vector θ, which satisfies the above equation, is called *TD(0)* solution—also this is the value found by LSTD(0).

In general, the ***TD-solution*** refers to the fixed-point of the expected TD update (e.g., Eq. 2.11). But under off-policy training, if this fixed-point exists, it may not be stable. In other words, if θ does satisfy in Equation (2.11), then the TD(0) algorithm in expectation may cause it to move away and eventually diverge to infinity.

Equation (2.11), gives us the TD-solution in the parameter space. In the value function space, the TD-solution, θ, satisfies

$$V_\theta = \Pi T V_\theta, \tag{2.14}$$

where $V_\theta = \Phi\theta \in \mathbb{R}^{|\mathcal{S}|}$, Φ is the matrix whose rows are the $\phi(s)^\top$ for a given row entry s, and Π is the projection operator to the linear space. The projection operator, Π, takes any value function v and projects it to the nearest value function representable by the function approximator:

$$\Pi v = V_\theta \text{ where } \theta = \arg\min_\theta \| V_\theta - v \|_\mu^2,$$

where μ is state-visitation probability distribution vector whose s^{th} component, $\mu(s)$, represents probability of visiting state s, and

$$\|v\|_\mu^2 = \sum_s \mu(s)v^2(s).$$

In a linear architecture, in which $V_\theta = \Phi\theta$, the projection operator is linear and independent of θ:

$$\Pi = \Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D, \tag{2.15}$$

where D is a diagonal matrix whose diagonal elements are $\mu(s)$. For a geometric picture of value function with linear structure see Figure 2.3.

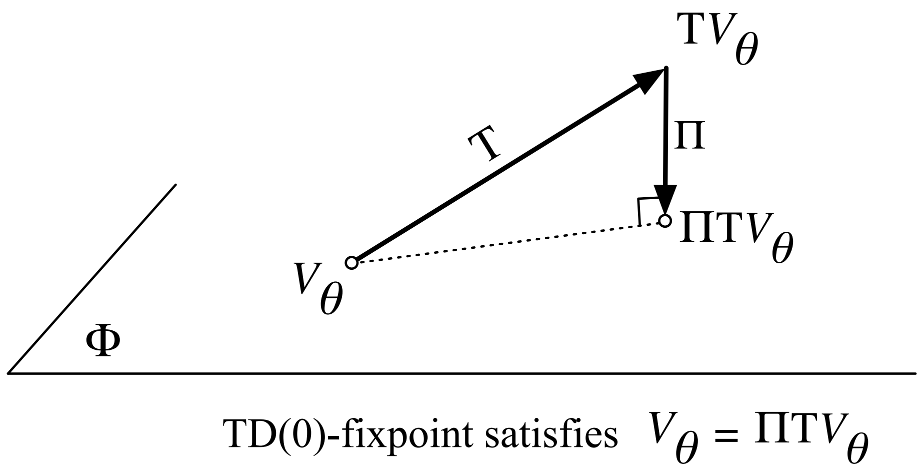

Figure 2.3: Geometric relationships between the approximate value functions, the the effect of Bellman and projection operators on value functions.

Unlike linear TD(0), nonlinear TD(0) can diverge under on-policy training (Tsitsiklis and Van Roy, 1997). Unfortunately, both linear and nonlinear TD(0) can diverge under off-policy training (Sutton and Barto, 1998; Tsitsiklis and Van Roy, 1997). The purpose of this thesis is to develop a new family of TD algorithms that are sound under general settings.

In the next section, I show how conventional TD methods, such as TD(0), are derived in conjunction with function approximation.

2.6 Derivation of TD(0) with function approximation

In this section, I briefly overview the derivation of TD(0) with function approximation. The purpose of this section it to show how linear/nonlinear TD(0) has been derived to get a better understanding why it may diverge. Particularly, I will show that linear/nonlinear TD(0) is not a true gradient-descent method, which (partially) might suggest why TD methods with function approximation are not robust for general settings.

To start, let's consider the following mean-square-error objective function

$$\mathrm{MSE}(\theta) = \mathbb{E}\Big[(V^\pi(S_t) - V_\theta(S_t))^2\Big] .$$

Following gradient-descent methods, the learning update can be obtained by adjusting the modifiable parameter θ along the steepest descent direction of the MSE objective function; that is, $\theta_{\mathrm{new}} - \theta_{\mathrm{old}} \propto -\frac{1}{2}\nabla \mathrm{MSE}(\theta)_{|\theta=\theta_{\mathrm{old}}}$, where

$$\begin{aligned} & -\frac{1}{2}\nabla \mathrm{MSE}(\theta_{\mathrm{old}}) \\ = \; & -\frac{1}{2}\nabla\left(\mathbb{E}\Big[(V^\pi(S_t) - V_\theta(S_t))^2\Big]\right)_{|\theta=\theta_{\mathrm{old}}} \\ = \; & \mathbb{E}\Big[(V^\pi(S_t) - V_{\theta_{\mathrm{old}}}(S_t))\, \nabla V_\theta(S_t)_{|\theta=\theta_{\mathrm{old}}}\Big] . \end{aligned} \tag{2.16}$$

In general, it is not practical to compute the above update term because: *1)* The target value, $V^\pi(S_t)$, is not known; *2)* We do not have access to model of the environment and therefore can not compute the expectation term.

We can get around the first problem by approximating the target value

$$V^\pi(s) \approx \mathbb{E}[R_{t+1} + \gamma V_\theta(S_{t+1}) \mid S_t = s, \pi] .$$

This is called *bootstrapping* step. To get around the second problem we use the theory of stochastic approximation; that is, at every time-step we conduct direct sampling from equation (2.16) and update the parameters along (stochastic) direction of

$$(V^\pi(S_t) - V_\theta(S_t))\, \nabla V_\theta(S_t).$$

Putting these all together, we get linear/nonlinear TD(0) algorithm:

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \nabla V_{\theta_t}(S_t), \tag{2.17}$$

where δ_t is one-step TD error (2.8).

2.7 The stability problem of linear/nonlinear TD(0)

In previous section, we overviewed the derivation of one of the simplest and popular TD method; that is TD(0), in conjunction with function approximation. In this section, we raise one of its most outstanding problems, that is, the stability problem.

In the next paragraph, we consider one of the well-known counterexamples, which shows the divergence of linear TD(0) and its approximate dynamic programming counterpart (updating by sweeping over the states instead of sampling).

Baird's Off-policy Counterexample Consider the 7-star version of the "star" counterexample (Baird, 1995; Sutton and Barto, 1998). The Markov decision process (MDP) is depicted in Fig. 2.4. The reward is zero in all transitions, thus the true value functions for any given policy is zero; for all states.

The behavior policy, in this example, chooses the solid line action with probability of $1/7$ and the dotted line action with probability of $6/7$. The goal is to learn the value of a target policy that chooses the solid line more often than the probability of $1/7$. In this example, the target policy choose the solid action with probability of 1.

The value functions are approximated linearly in the form of $V(i) = 2\theta(i) + \theta_0$, for $i \in \{1, ..., 6\}$, and $V(7) = \theta(7) + 2\theta_0$. Here, the discount factor is $\gamma = 0.99$. The TD-solution, in this example, is $\theta(i) = 0$, $i \in \{1, ..., 7\}$, and $\theta_0 = 0$. Both TD(0) and DP (with incremental updates), however, will diverge on this example; that is, their learning parameters will go to $\pm\infty$ as is illustrated in Fig. 2.5.

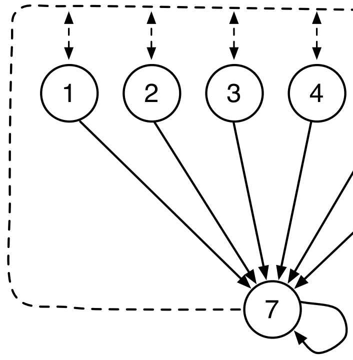

Figure 2.4: The Baird's 7-star MDP. Every transition in this MDP receives zero reward. Each state, has two actions, represented by solid line and dotted line. Solid line action only make transition to state state 7, while dotted line action uniformly make transition to one of states 1-6 with probability of $1/6$.

Now, we turn into the stability issues of TD methods in conjunction with nonlinear function approximation. Despite linear TD(0), nonlinear TD(0) can become unstable and diverge even under on-policy training. The spiral counterexample, due to Tsitsiklis and Van Roy (1997), shows divergence of nonlinear TD(0) under on-policy training.

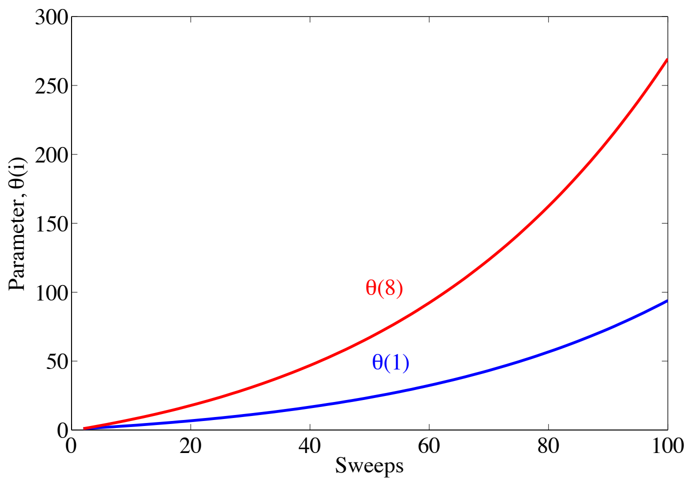

Figure 2.5: The learning parameters in Baird's counterexample diverge to infinity. The parameters are updated according to expected TD(0) update (dynamic programming style).

Spiral Counterexample Consider the Markov chain with 3 states as is shown in the left panel of Fig. 2.6. All state-state transitions are with probability of $1/2$, the reward is always zero, and the discount factor is $\gamma = 0.9$. The parametrized (approximate) value function has a scalar parameter, θ, and takes the nonlinear spiral form

$$V_\theta(s) = \Big(a(s)\cos{(\hat{\lambda}\theta)} - b(s)\sin{(\hat{\lambda}\theta)}\Big)\, e^{\epsilon\theta}.$$

The true value function is $V = (0,0,0)^\top$, which is achieved as $\theta \to -\infty$. The right panel of the figure, demonstrates the value functions for each state; each axis corresponds to value function for a given state. Here, we used $V_0 = (100,-70,-30)^\top$, $a = V_0$, $b = (23.094, -98.15, 75.056)^\top$, $\hat{\lambda} = 0.866$ and $\epsilon = 0.05$. As is illustrated in Fig. 2.6(c), the learning parameter θ diverges using nonlinear TD(0). Note, here we have used expected TD(0) update in the plot to illustrate how θ parameter evolves under TD learning.

Why is TD(0) with linear/nonlinear function approximation sometimes unstable? To address this question, first we need to look at the way the algorithm is derived. Unlike supervised learning methods, which use a mean-square-error objective function, in RL the objective is to estimate value functions that satisfy in Bellman equation. This would be straightforward to do if we use tabular representation. However, for the case of function approximation, it is not clear what is the underling equation of approximate value functions. In other words, it is not straight forward to do function approximation.

TD methods with function approximation are proposed as a way of conducting this approx-

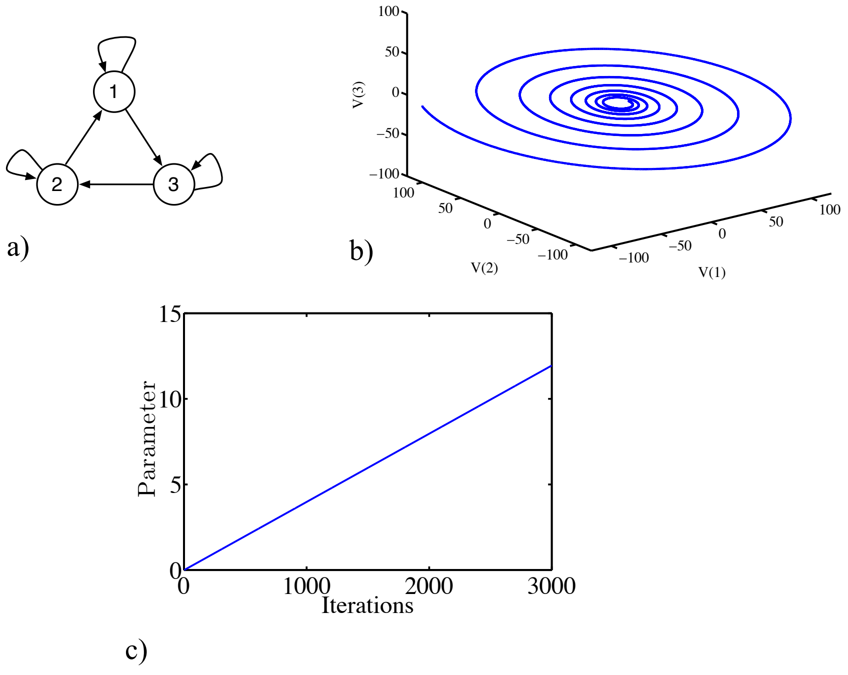

Figure 2.6: Spiral counterexample, which shows the divergence of nonlinear TD(0) under on-policy training. Panel a) shows the MDP problem and panel b) shows how the value functions evolve. Panel c) shows the learning parameter diverges to infinity. The parameter is updated according to expected TD(0) update (approximate dynamic programming style).

imation. In previous section, we showed how linear/nonlinear TD(0) originally has been derived. To do this, we started with the idea of updating the learning parameters along the gradient-descent direction of the mean-square error objective function, but with an approximation step , $V^{\pi}(s) \approx \mathbb{E}[R_{t+1} + \gamma V_\theta(S_{t+1}) \mid S_t = s]$. Thus, the resulting algorithm would not be true stochastic gradient-descent method. As a result, TD(0) with function approximation may diverge.

A good way to see why linear/nonlinear TD(0) is not a true gradient-descent method is to show that its update can not be derived by taking gradient of any function (Barnard, 1993). To do this, let's consider linear TD(0) with the update term, $\delta(\theta)\phi$, where $\delta(\theta) = R + \gamma\theta^\top\phi' - \theta^\top\phi$, $\phi \equiv \phi(S_t)$, $\phi' \equiv \phi(S_{t+1})$, where t represents time-step.

Now, let's assume that there exists a function $J(\theta)$ whose gradient is the TD(0) update, $\delta(\theta)\phi$; that is, $\nabla J(\theta) = \delta(\theta)\phi$. Thus, the j^{th} element of $\nabla J(\theta)$ is

$$\frac{\partial J(\theta)}{\partial \theta_j} = \delta(\theta)\phi_j.$$

Now, if we take another partial derivative with respect to the i^{th} component, we get

$$\frac{\partial^2 J(\theta)}{\partial\theta_j \partial\theta_j} = (\gamma\phi_i' - \phi_i)\phi_j,$$

and consequently we conclude

$$\frac{\partial^2 J(\theta)}{\partial\theta_i \partial\theta_j} \neq \frac{\partial^2 J(\theta)}{\partial\theta_j \partial\theta_i}. \tag{2.18}$$

This is a contradiction, because the second derivative of a differentiable function is independent of the order of derivatives. This derivation, however, shows that the second derivative of function J is not symmetric with respect to the order of derivative. As such, we conclude linear TD(0) update is not a gradient of any function!

Note, the point of this analysis is not to show that as a result of the inequality (2.18) linear TD(0) can diverge, but rather to show that linear TD(0) is not a true gradient-descent method. In fact, pseudo gradient methods, such as on-policy linear TD(0), can have convergence guarantee.

2.8 Residual gradient method

The residual gradient (RG) method (Baird, 1995) has been proposed as an alternative method to TD(0) with function approximation. The RG method updates the modifiable parameter θ along the gradient-descent direction of the following objective function:

$$\begin{aligned} J(\theta) &= \|TV_\theta - V_\theta\|_\mu^2 \\ &= \mathbb{E}\big[(\mathbb{E}[\delta_t(\theta)|S_t])^2\big], \end{aligned} \tag{2.19}$$

where $\delta_t(\theta) = R_{t+1} + \gamma V_\theta(S_{t+1}) - V_\theta(S_t)$ is TD error, and

$$\begin{aligned} -\frac{1}{2}\nabla J(\theta) &= \mathbb{E}[\mathbb{E}[\delta_t(\theta) \mid S_t]\,\mathbb{E}[\nabla V_\theta(S_t) - \gamma\nabla V_\theta(S_{t+1}) \mid S_t]] \\ &= \mathbb{E}[\delta_t(\theta)\nabla V_\theta(S_t)] - \gamma\mathbb{E}\Big[\mathbb{E}[\delta_t(\theta)|S_t]\,\mathbb{E}[\nabla V_\theta(S_{t+1}) \mid S_t]\Big]. \end{aligned} \tag{2.20}$$

Because the RG method is derived based on (true) gradient descent, it has stability guarantees. However, one problem with the RG method is that direct sampling is not straightforward from the second term of the above equation. That is, given state, S_t, two independent samples for the next state are required in order to do unbiased stochastic gradient descent. This is also known as the RG method double sampling problem. The only case that this would not be an issue is when the system is deterministic or the model of the environment is known, which is not always the case.

As a result, Baird (1995) recommended using only one sample for the next state yielding the following variant of the RG method:

$$\Delta\theta_t = \alpha_t \delta_t (\nabla V_{\theta_t}(S_t) - \gamma \nabla V_{\theta_t}(S_{t+1})), \tag{2.21}$$

where α_t is positive step-size parameter. The RG method with one sample for the next state, can be viewed as true gradient-descent in expected squared TD error objective function, $\mathbb{E}\big[\delta_t^2(\theta)\big]$.

A convergence guarantee can be assured for both of the RG algorithms, however, RG solutions have been found to be inferior to the TD solution, as is exemplified in the next chapter (see also Sutton et al., 2009).

2.9 Conclusion

In this chapter, I gave a background review of some of the most popular TD methods, such as TD(λ), Q-learning and Sarsa. First, I presented the algorithms in their tabular form and then combined them with function approximation. I showed linear/nonlinear TD algorithms are not true gradient descent and as such the condition under which they converge is narrow, and they can diverge under off-policy training (e.g., Baird, 1995).

In the next section we study several possible objective functions that can be used in RL (as well as approximate dynamic programming). We will describe the merits of the one that leads to the TD-solution, and based on this objective function we build all of our future algorithms.

Chapter 3

Objective Function for Temporal-Difference Learning

An objective function is some function of the modifiable parameter θ that we seek to minimize by updating θ. In (stochastic) gradient-descent, the updates to θ are proportional to the negative (sample) gradient of the objective function with respect to θ. In standard RL, the objective is to find solution that satisfies Bellman equation. However, in the case of function approximation, it is not clear how to combine the Bellman equation with value function approximation.

In this chapter we seek an objective function (for the case of policy evaluation), whose minimum value provides a reasonable solution for approximate value functions. We propose a Bellman-error objective function in which the TD-solution is its optimum solution. We conclude that in fact this objective function is fairly reliable and thus the rest of our analysis will be conducted based on it.

3.1 Several potential objective functions

In this section[1], first we consider several objective functions that might be useful for approximate dynamic programming as well as temporal-difference learning methods. Then in the next section, we study the reliability of each and eventually choose the one that is most reliable.

[1]This section is based on the following paper: Sutton, R. S., Maei, H. R, Precup, D., Bhatnagar, S., Silver, D., Szepesvári, Cs. & Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 993–1000. Omnipress.

Mean-square-error (MSE) One natural choice for the objective function might be the mean-squared-error (MSE) between the approximate value function V_θ and the true value function V^π, that is,

$$\begin{aligned} \mathrm{MSE}(\theta) &= \sum_s \mu(s) \left(V_\theta(s) - V^\pi(s)\right)^2 \\ &\overset{\text{def}}{=} \parallel V_\theta - V^\pi \parallel_\mu^2, \end{aligned}$$

where V_θ and V^π are viewed as vectors with one element for each state, and the norm $\parallel v \parallel_\mu^2 = \sum_s \mu(s) v^2(s)$. The V^π is the true value function vector, and under the MDP condition it satisfies the Bellman equation (2.2).

Mean-square Bellman-error (MSBE): A seemingly natural measure of how closely the approximation V_θ satisfies the Bellman equation is the *mean-square Bellman error*:

$$\mathrm{MSBE}(\theta) = \parallel V_\theta - TV_\theta \parallel_\mu^2, \tag{3.1}$$

where T is Bellman operator (for simplicity we have dropped the superscript π from T operator).

This is the objective function used by the most important prior effort to develop gradient-descent algorithms (Baird, 1995; Baird, 1999). However, most popular temporal-difference algorithms, including TD(0) with function approximation, do not converge to the minimum of the MSBE. To understand this, note that the Bellman operator follows the underlying state dynamics of the Markov chain, irrespective of the structure of the function approximator. As a result, TV_θ will typically not be representable as V_θ for any θ.

The residual gradient (RG) method, in fact was proposed to find the optimum solution of this objective function, however, due to some technical issues—e.g., the requirement for having two independent next-state, given the current state— it has not been practical in a wide range of real-world applications. Thus, instead, it has been proposed to use only one sample for the next state, given the current state. However, the solution (called RG solution) is considered inferior to TD-solution.

Mean-square TD-error (MSTDE): Another choice for objective function would be to minimize mean-square TD-error (MSTDE); that is,

$$\mathrm{MSTDE}(\theta) = \mathbb{E}\Big[\delta_t(\theta)^2\Big], \tag{3.2}$$

where $\delta_t(\theta) = r_{t+1} + \gamma V_\theta(S_{t+1}) - V_\theta(S_t)$. The major problem with this objective function is its inferior results. For example, for the case of tabular representations, in general, its optimum solution does not satisfy the Bellman equation. The RG solution, is the optimum solution for the MSTDE objective function.

Mean-square projected Bellman-error (MSPBE): In the previous chapter, I showed that the TD-solution for linear TD(0) satisfies $V_\theta = \Pi T V_\theta$. Thus, another choice for the objective function would be to take the mean-square *projected* Bellman-error (MSPBE) objective function:

$$J(\theta) \quad = \quad \| V_\theta - \Pi T V_\theta \|_\mu^2 \; . \tag{3.3}$$

Fig. 3.1 shows the geometric relationship between MSPBE ($J(\theta)$) and MSBE . Although

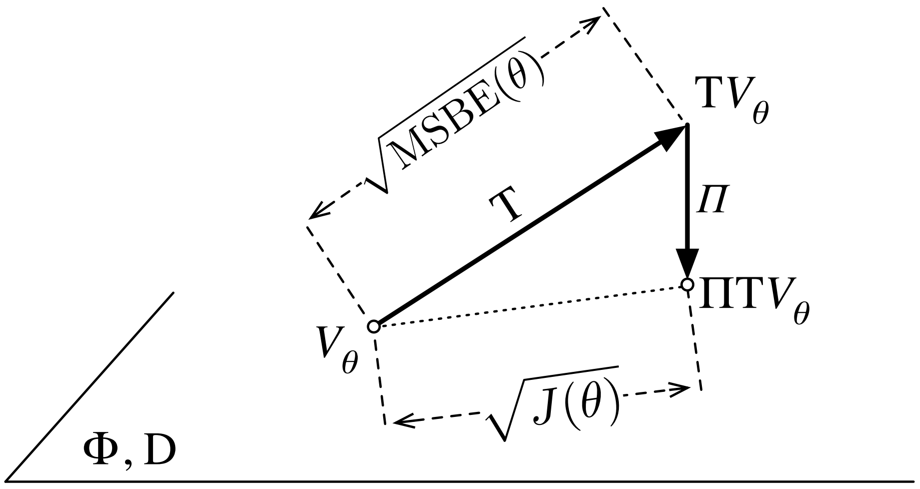

Figure 3.1: Geometric relationships between the square roots of the two Bellman-error objective functions. Note, the reason we have considered D as part of the function space is that projected operator Π is weighted according to D.

many previous works have highlighted the goal of achieving the TD solution (2.14). Our present work seems to be the first to focus on minimizing MSPBE as an objective function to be minimized through gradient-descent method. The idea of minimizing MSPBE also was proposed in Antos, Szepesvári and Munos (2007, p. 100, 2008), which discusses viewing LSTD solution as minimizing the MSPBE. Further insight into the difference between the two Bellman-error objective functions can be gained by considering the episodic example in Fig. 3.2.

3.2 Which objective function to choose?

To gain further insight into the difference between our candidate objective functions, let's consider the episodic example illustrated in Fig. 3.2, and compare the solutions obtained from the proposed objective functions.

In the left and middle panels, episodes begin in state A. Then state A makes a transition to B or to C with probability of $1/2$ before proceeding to termination with a reward of 1 or 0 (all other transitions have zero reward). We consider the tabular representation for the states, thus, we expect to get the true solution which can be obtained from Bellman equation.

In the right panel, called A-split example, the state A now is split into two states, $\mathsf{A1}$ and $\mathsf{A2}$ that share the same feature representation; they look the same and must be given the same approximate value. Thus, the feature vector for both $\mathsf{A1}$ and $\mathsf{A2}$ is $\phi(\mathsf{A1}) = \phi(\mathsf{A2}) = (1, 0, 0)^\top$, also we have $\phi(\mathsf{B}) = (0, 1, 0)^\top$ and $\phi(\mathsf{C}) = (0, 0, 1)^\top$. Trajectories start in one of the two A states each with 50% probability, then proceed deterministically either to B and 1, or to C and 0.

First, we consider the following notation: Let random samples be in the form of triple (S, R, S') with corresponding state feature vectors, $\phi(S)$ and $\phi(S')$ and R denotes reward along the transition. TD error for this transition is $\delta(\theta) = R + \gamma\theta^\top\phi(S') - \theta^\top\phi(S)$. Here, we compare the following four solutions:

- The exact solution obtained from the Bellman equation.
- The TD solution, θ, which satisfies $\mathbb{E}[\delta(\theta)\phi] = 0$, and also minimizes the MSPBE objective function (3.3).
- The solution, which minimizes the MSBE objective function (2.19):

$$\begin{aligned} \text{MSBE}(\theta) &= \|TV_\theta - V_\theta\|_\mu^2 \\ &= \mathbb{E}\big[(\mathbb{E}[\delta(\theta)|S])^2\big]\,. \end{aligned}$$

- The RG solution, which minimizes the following objective function:

$$\text{MSTDE}(\theta) = \mathbb{E}\Big[\delta(\theta)^2\Big]\,,$$

  where was introduced in Equation (3.2).

In the left panel, we can see states B and C are given values of 1 and 0 respectively according

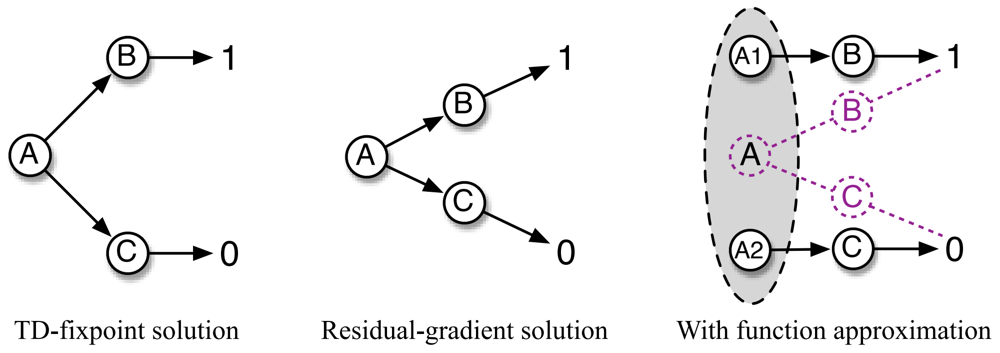

Figure 3.2: In the left and middle panels, episodes begin in state A then transition either to B or to C with equal probability before proceeding to termination with a reward of 1 or 0 (all other transitions have zero reward). The vertical positions of the states represent their values according to the TD solution (left panel) and according to the residual-gradient (RG) solution (middle panel; Baird 1995, 1999). State A, for example, has height midway between 0 and 1 in both solutions, corresponding to its correct value of $\frac{1}{2}$ (because episodes starting in A end half the time with a total reward of 1 and half the time with a total reward of 0, and $\gamma = 1$).

to TD solution, which is the exact solution obtained from the Bellman equation. However, in the RG solution they are given the values $\frac{3}{4}$ and $\frac{1}{4}$. The 1,0 values are correct in that these states are always followed by these rewards, but they result in large TD errors, of $\delta = \pm\frac{1}{2}$, on transitions out of A.

The RG solution has smaller TD errors, of $\delta = \pm\frac{1}{4}$, on all of its transitions, resulting in a smaller mean-square TD error *per episode* of $\frac{1}{4}^2 \times 2 = \frac{1}{8}$ as compared to $\frac{1}{2}^2 = \frac{1}{4}$ for the TD solution. That is, the RG solution (with one sample for the next state) splits the TD error over two transitions to minimize squared TD error overall.

The key difference here is that, from A, the squared TD error tends to be large but the expected TD error (the Bellman error) tends to be zero (as long as the B and C values are distributed symmetrically around $\frac{1}{2}$).

The TD solution 1,0 is in fact the minimum of MSBE on this problem, and this has led to the widespread belief that the MSBE solves this problem. However, this is not the case in general; once function approximation, including unobservable states, is introduced, the MSBE and MSPBE solutions differ, and the $\frac{3}{4}$,$\frac{1}{4}$ solution may reappear.

An example of this is shown in the right panel of Figure 3.2. From the observable data, this example looks just like the previous, except now taking multiple samples is no help because the system is deterministic, and they will all be the same. Now the $\frac{3}{4}$,$\frac{1}{4}$ solution minimizes not just the squared TD error, but the MSBE as well; only the MSPBE criterion puts the minimum at the $1, 0$ solution.

The MSBE objective causes function approximation resources to be expended trying to reduce the Bellman error associated with A1 and A2, whereas the MSPBE objective takes into account that their approximated values will ultimately be projected onto the same value function.

Scherrer (2010) has conducted analysis on the reliability of both the TD-solution and the RG-solution. He also provides an oblique projection framework that can describe the solutions obtained from the TD(0) algorithm and the RG method, which is limited to one-step TD methods. However, note that the solution of TD methods with linear/nonlinear function approximation changes in conjunction with eligibility traces. Thus, selecting a proper value for bootstrapping parameter, λ, can provide a quality solution using TD(λ).

However, we think, A-split examples provides some insights that tell us there is a fundamental problem with MSBE objective function when states are invisible—this is also inherent in the notion of function approximation. It is interesting to note, function approximation is related to partial observability of states. That is, one reason we do function approximation, is because the state-space is too large and we can not observe all of them.

The A-split example demonstrates that the MSBE result could be inferior to the TD-solution, however, this is not the only reason we abandon the MSBE objective function here. One major issue is double-sampling restriction for its stochastic gradient descent direction. The second issue is the fact that, in large scale problems, all the states are not visible. Thus, we use function approximation.

One way to fix this problem (and also the problem with MSBE), for the case of function approximation, is to condition the expectation term with features and not the actual states. Thus we define the following feature-based MSBE objective function:

$$\tilde{J}(\theta) \quad = \quad \mathbb{E}\big[(\mathbb{E}[\delta(\theta)|\phi(S)])^2\big]\,.$$

Now, following the derivations in Section 2.8 (see Eq. 2.20), we get

$$\begin{aligned} -\frac{1}{2}\nabla \hat{J}(\theta) \quad &= \quad \mathbb{E}[\delta(\theta)\nabla V_\theta(S)] - \gamma\mathbb{E}\big[\mathbb{E}[\delta(\theta)|S]\,\mathbb{E}\big[\nabla V_\theta(S') \mid \phi(S)\big]\big] \\ &= \quad \mathbb{E}[\delta(\theta)\phi(S)] - \gamma\mathbb{E}\big[\mathbb{E}[\delta(\theta)|S]\,\mathbb{E}\big[\phi(S') \mid \phi(S)\big]\big]\,, \end{aligned}$$

where we have used linear function approximation: $V_\theta(S) = \theta^\top\phi(S)$. By direct sampling, we get the following update

$$\theta \leftarrow \theta + \alpha \left[\delta(\theta)\phi(S) - \gamma \mathbb{E}[\delta(\theta)|S]\,\phi(S')\right].$$

Now, because we are using linear function approximation, it makes sense to represent the value of $\mathbb{E}[\delta(\theta)|S]$ with a measure in the feature space, that is, $\mathbb{E}[\delta(\theta)|S] \approx w(\theta)^\top \phi(S)$. It turns out that the best linear fit (using mean-square error criteria) yields,

$$w(\theta) = \mathbb{E}[\phi(S)\phi(S)]^{-1}\,\mathbb{E}[\delta(\theta)\phi(S)].$$

As a result, the ultimate update becomes

$$\theta \leftarrow \theta + \alpha \left[\delta(\theta)\phi(S) - \gamma \left(w(\theta)^\top \phi(S)\right)\phi(S')\right], \quad w(\theta) = \mathbb{E}[\phi(S)\phi(S)]^{-1}\,\mathbb{E}[\delta(\theta)\phi(S)].$$

Interestingly, the direction of the above update is along the stochastic gradient-descent direction in the MSPBE objective, which eventually leads to TD solution (see also Chapter 5, particularly, the TDC algorithm, for further details).

Finally, we close this discussion of objective functions by giving the following function, which does not seem to have a ready geometric interpretation, however, its minimum satisfies the TD solution. Here we call it the *norm of the expected TD update*:

$$\text{NEU}(\theta) = \mathbb{E}[\delta(\theta)\phi]^\top\,\mathbb{E}[\delta(\theta)\phi], \tag{3.4}$$

which results in TD solution.

3.3 Conclusion

In this chapter, we introduced several potential objective functions for TD learning and showed the pros and cons of each of them. Finally, we noted that the MSPBE objective function seems to be more reliable than the MSBE objective function. Before deriving our gradient-TD algorithms based on the MSPBE (and NEU) objective function, first we formulate the problem of off-policy learning in the next chapter.

Chapter 4

Off-Policy Formulation of Temporal-Difference Learning

One of the key features of TD learning is its ability to learn from incomplete sequences without waiting for the outcome. This key feature, allows TD methods to learn from single state-state transitions (smallest fragment of experience). It turns out, we can use this unique property to do off-policy learning.

In this chapter, we provide an off-policy formulation for temporal-difference learning based on sub-sampling[1] from data generated according to the agent behavior policy, that is, from trajectories. In reality, however, we would like to use all the data. As such, we show how to use importance-sampling technique, which allows us to use all of the samples.

4.1 Sub-sampling and i.i.d. formulation of TD learning

In this section, we formulate the off-policy prediction problem for one-step temporal-difference learning. For simplicity, we consider off-policy data in the form of independent, identically distributed (i.i.d.) samples. Our goal is to learn from off-policy data using conventional TD methods, such as TD(0) with linear function approximation.

We start by considering the standard reinforcement learning framework. In this framework, the environment and the agent behavior policy, π_b , together generate a stream of states, actions and rewards, $S_1, A_1, R_1, S_2, A_2, R_2, \ldots$, which we can break into causally related 4-tuples, (S_1, A_1, R_1, S'_1), (S_2, A_2, R_2, S'_2), $\ldots$, where $S'_t = S_{t+1}$. The generated data is called on-policy data and as such forms a trajectory. For some tuples, the action will match

[1]Some parts of this section, including the sub-sampling and i.i.d. formulation of TD learning are based on the following paper: Sutton, R. S., Szepesvári, Cs., Maei, H. R. (2009). A convergent $O(n)$ algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in Neural Information Processing Systems 21*. MIT Press.

what the target policy (the policy that we are learning about) would do in that state, and for others it will not. We can discard all of the latter as not relevant to the target policy. For the former, we can discard the action because it can be determined from the state via the target policy. With a slight abuse of notation, let S_k denote the kth state in which an on-policy action was taken, and let R_k and S'_k denote the associated reward and next state. The k^{th} on-policy transition, denoted (S_k, R_k, S'_k), is a triple consisting of the starting state of the transition, the reward on the transition, and the ending state of the transition. The corresponding data available to the learning algorithm, say for the case of linear function approximation, is the triple (ϕ_k, R_k, ϕ'_k), where $\phi_k = \phi(S_k)$ and $\phi'_k = \phi(S'_k)$. Now, we have a bag of data, which looks like a collection of snippets instead of a trajectory. Now we can consider that this bag of data has a distribution and thus we can remove the indices and talk about a single tuple of random variables (ϕ, R, ϕ').

In the i.i.d. formulation, the states S_k are generated independently and identically distributed according to an arbitrary probability distribution, μ (μ is a vector whose s^{th} element is $\mu(s)$), which also refers to starting state distribution. From each S_k, a corresponding S'_k is generated according to the on-policy state-transition matrix, P , and a corresponding R_k is generated according to an arbitrary bounded distribution. The final i.i.d. data sequence, from which an approximate value function is to be learned, is then the sequence $(\phi(S_k), R_k, \phi(S'_k))$, for $k = 1, 2, ...$ Further, because each sample is i.i.d., we can remove the indices and talk about a single tuple of random variables (ϕ, R, ϕ').

It is worth to mention that if the bag of data is collected by trajectory following; that is, on-policy data, then the underlying state distribution, μ, and state-state transition probability distribution P are linked according to $\mu^\top P = \mu^\top$. This constraint, is the main reason behind the convergence of TD methods with linear function approximation. If for any reason (such as off-policy scenario), μ and P are not linked according to this linear equation, the convergence can't be guaranteed.

4.2 Importance-weighting formulation

In the previous section, for the sake of simplicity, we introduced sub-sampling and i.i.d. formulation of TD learning. However, in reality, we would like to use all of data sequence. In addition, this would allow us to conduct moment-to-moment (online) predictions without throwing away some data.

Now, consider that data sequence is generated according to the behavior policy $\pi_b : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Let t^{th} sample be in the form of triple (S_t, R_{t+1}, S_{t+1}). Let the target policy

π and behavior policy π_b be stochastic. According to sub-sampling formulation we get a bag of matched samples (possibly infinite) in the form of triples $(S_k, R_k, S'_k)_{k\geq 0}$, which the notation was introduced in the previous section.

Now the linear TD(0) will be:

$$\theta_{k+1} = \theta_k + \alpha_k \delta_k \phi_k,$$

where $\delta_k = R_k + \gamma\theta_k^\top \phi'_k - \theta_k^\top \phi_k$. Now let $\delta_k(\theta) = R_k + \gamma\theta^\top \phi'_k - \theta^\top \phi_k$. The TD solution, θ, $\mathbb{E}[\delta_k(\theta)\phi_k] = 0$.

Now we ask: how can we sample from the expected TD update, $\mathbb{E}[\delta_k(\theta)\phi_k]$, while the data is generated according to the agent's behavior policy. Before answering this question, we adopt the following notations: Let

$$\delta(\theta; s, a, s') = r(s, a, s') + \gamma V_\theta(s') - V_\theta(s). \tag{4.1}$$

Because $S \sim \mu(.)$ and $S' \sim P^\pi(.|S)$, we have

$$\begin{aligned} \mathbb{E}[\delta_k(\theta)\phi_k] &= \sum_s \mu(s)\pi(a|s)P(s'|s,a)\delta(\theta; s, a, s')\phi(s) \\ &\overset{\text{def}}{=} \mathcal{P}_\mu^\pi \delta(\theta)\phi, \end{aligned} \tag{4.2}$$

where $\mathcal{P}_\mu^\pi$ is an operator, $\delta(\theta) = R + \gamma V_\theta(S') - V_\theta(S)$, and $\phi = \phi(S)$.

Thus, off-policy TD-solution, θ, satisfies

$$\mathcal{P}_\mu^\pi \delta(\theta)\phi = 0. \tag{4.3}$$

The following Lemma will help us to find the off-policy TD-solution, as well as off-policy TD(0) update, in a mechanistic way, without going through sub-sampling scenario.

Lemma 1. *(Importance-weighting for off-policy TD) Consider that data $(S_t, R_{t+1}, S_t)_{t\geq 0}$ is generated according to a stationary behavior policy π_b, and let the stationary target policy be π. Then from Equation (4.2) we have*

$$\mathcal{P}_\mu^\pi \delta(\theta)\phi = \mathbb{E}[\rho_t \delta_t(\theta)\phi_t], \tag{4.4}$$

where $\rho_t \equiv \pi(A_t \mid S_t)/\pi_b(A_t \mid S_t)$, and $\delta_t(\theta) = R_{t+1} + \gamma\theta^\top\phi_{t+1} - \theta^\top\phi_t$ is TD error for a given sample (S_t, R_{t+1}, S_{t+1}), $\phi_t \equiv \phi(S_t)$, and $\phi_{t+1} \equiv \phi(S_{t+1})$.

Proof. Because the two policies are stationary and fixed, thus naturally the expectation term is well defined. Thus, we have

$$\begin{aligned}
&\mathcal{P}_\mu^\pi \delta(\theta)\phi \\
&= \sum_{s,a,s'} \mu(s)\pi(a \mid s)P(s' \mid s,a)\delta(\theta; s,a,s')\phi(s) \\
&= \sum_{s,a,s'} \mu(s)\frac{\pi(a, \mid s)}{\pi_b(a \mid s)}\pi_b(a \mid s)P(s' \mid s,a)\delta(\theta; s,a,s')\phi(s) \\
&= \sum_{s,a,s'} \mu(s)\pi_b(a \mid s)P(s' \mid s,a)\frac{\pi(a, \mid s)}{\pi_b(a \mid s)}\delta(\theta; s,a,s')\phi(s) \\
&= \mathbb{E}[\rho_t\delta_t(\theta)\phi_t]\,.
\end{aligned}$$

Here, $P^\pi(s' \mid s) = \pi(a \mid s)P(s' \mid s,a)$— note $P(s' \mid s,a)$ depends on the environment and not the agent. □

Note, the state distribution, μ, refers to the actual state distribution of data generated according to agent's behavior policy.

From Lemma 1, by a direct sampling, we get the following linear TD(0) algorithm for off-policy learning:

$$\theta_{t+1} = \theta_t + \alpha_t\rho_t\delta_t\phi_t,$$

where $\delta_t \equiv \delta_t(\theta_t)$. The above update weights the (on-policy) linear TD(0) update by the likelihood of action taken by target policy (as opposed to behavior policy). Here, we have assumed that there is always a chance for an action to be taken according to behavior policy; that is, the importance-weight (likelihood ratio) is bounded.

However, the above off-policy learning algorithm is subject to the possibility of divergence. In the next chapter, we propose a new gradient TD approach that provides a building block for developing convergent TD methods for general settings.

4.3 Conclusion

In this chapter we have provided a general formulation for off-policy learning, which is consistent to off-policy learning methods, such as Q-learning. In this formulation, the target policy state-state transitions are incorporated into the learning, while state-visitation distributions are due to the agent's behavior. In order to use all the data efficiently, we have also incorporated an importance-weighting technique into the updates.

Chapter 5

Gradient Temporal-Difference Learning with Linear Function Approximation

This chapter[1] provides the core ideas, and theoretical results behind the gradient-TD algorithms for the case of linear function approximation. Here, we explore the development of true stochastic gradient-descent algorithms for temporal-difference learning with linear function approximation, building on the work of Baird (1995; 1999).

Particularly, we introduce three new TD algorithms compatible with both linear function approximation and off-policy training, and whose complexity scales only linearly in the size of the function approximator. The first algorithm, GTD, estimates the expected update vector of the TD(0) algorithm and performs stochastic gradient descent on its L2 norm; that is, norm of the expected TD update, also called NEU (see Chapter 3).

In Section 5.3, we prove that GTD is stable and convergent to the TD-solution under the usual stochastic approximation conditions and i.i.d. assumption on off-policy data. The second and third algorithms, GTD2 and TDC (TD with gradient correction term), are derived and proved convergent just as GTD was, but use the projected Bellman-error objective function (see Chapter 3) and converges significantly faster (but still not as fast as conventional TD).

[1]This chapter is based on the following papers:

- Sutton, R. S., Szepesvári, Cs., Maei, H. R. (2009). A convergent $O(n)$ algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in Neural Information Processing Systems 21*. MIT Press.
- Sutton, R. S., Maei, H. R, Precup, D., Bhatnagar, S., Silver, D., Szepesvári, Cs. & Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 993–1000. Omnipress.

In our experiments on small test problems, the learning rate of these algorithms were compared to TD(0). To get further insights about the performance of these new algorithms on large-scale problems, David Silver implemented the gradient TD algorithms in a Computer Go application with a million features (see Sutton, Maei, et al. 2009).

Our empirical results shows TDC's rate of convergence is, fairly, faster than GTD2 and GTD, but it can be still slower than TD(0) on the class of on-policy problems, in which linear TD(0) is sound. All of these new linear algorithms extend linear TD(0) to off-policy learning with convergence guarantee, while doubling computational requirements.

Our empirical results suggest TDC is the most efficient algorithm among GTD and GTD2. Thus, in forthcoming chapters, we will build our new algorithms based on TDC.

5.1 Derivation of the GTD algorithm

We next present the idea and gradient-descent derivation leading to the GTD algorithm. As discussed in previous chapter (see off-policy i.i.d. formulation in Section 4.1), we consider i.i.d. samples $(S_k, R_k, S'_k)_{k\geq 0}$, consisting of the starting state of the transition, the reward on the transition, and the ending state of the transition. For the case of linear function approximation, the k^{th} corresponds to the triple (ϕ_k, R_k, ϕ'_k), where $\phi_k = \phi(S_k)$ and $\phi'_k = \phi(S'_k)$.

Note, the starting state distribution is μ, the state-state transition probability is P, and the vector $\mathbb{E}[\delta\phi]$ is expected TD update that can be viewed as an error in the current solution θ. The vector should be zero, so its norm is a measure of how far we are away from the TD solution. A distinctive feature of our gradient-descent analysis of temporal-difference learning is that we use as our objective function the L2 norm of this vector:

$$\text{NEU}(\theta) = \mathbb{E}[\delta(\theta)\phi]^\top \, \mathbb{E}[\delta(\theta)\phi] \,,$$

where was introduced in Equation (3.4).

This objective function is quadratic and unimodal; its minimum value of 0 is achieved when

$\mathbb{E}[\delta\phi] = 0$. The gradient-descent direction of this objective function is

$$\begin{aligned}
-\frac{1}{2}\nabla \mathrm{NEU}(\theta) &= -(\nabla \mathbb{E}\Big[\delta(\theta)\phi^\top\Big])\mathbb{E}[\delta(\theta)\phi] \\
&= -\mathbb{E}\Big[\nabla\delta(\theta)\phi^\top\Big]\,\mathbb{E}[\delta(\theta)\phi] \\
&= -\mathbb{E}\Big[(\gamma\phi' - \phi)\phi^\top\Big]\,\mathbb{E}[\delta(\theta)\phi] \\
&= \mathbb{E}\Big[(\phi - \gamma\phi')\phi^\top\Big]\,\mathbb{E}[\delta(\theta)\phi]\,. \qquad (5.1)
\end{aligned}$$

We would like to take a stochastic gradient-descent approach, in which a small change is made on each sample in such a way that the expected update is the direction opposite to the gradient. This is straightforward if the gradient can be written as a single expected value, but here we have a product of two expected values. One cannot sample both of them because the sample product will be biased by their correlation. However, one could store a long-term, quasi-stationary estimate of either of the expectations and then sample the other. The question is, which expectation should be estimated and stored, and which should be sampled? Both ways seem to lead us to different algorithms.

First let us consider the algorithm obtained by forming and storing a separate estimate of the first expectation; that is, of the matrix $A = \mathbb{E}\big[\phi(\phi - \gamma\phi')^\top\big]$. This matrix is straightforward to estimate from experience as a simple arithmetic average of all previously observed sample outer products $\phi(\phi - \gamma\phi')^\top$. Note that A is a stationary statistic in any fixed-policy policy-evaluation problem; it does not depend on θ and would not need to be re-estimated if θ were to change. Let A_k be the estimate of A after observing the first $k+1$ samples, $(\phi_0, R_0, \phi'_0), \cdots, (\phi_k, R_k, \phi'_k)$. Then this algorithm is defined by

$$A_k = \frac{1}{k}\sum_{i=0}^{k} \phi_i \left(\phi_i - \gamma\phi'_i\right)^\top,$$

along with the gradient descent rule:

$$\theta_{k+1} = \theta_k + \alpha_k {A_k}^\top \left(\delta_k\phi_k\right), \quad k \geq 0,$$

where θ_0 is arbitrary, $\delta_k = R_k + \gamma\theta_k^\top\phi'_k - \theta_k\phi_k$, and $(\alpha_k)_{k\geq 0}$ is a series of step-size parameters, possibly decreasing over time. We do not consider the above algorithm further here because it requires $O(d^2)$ memory and computation per-time-step.

The second path to a stochastic-approximation algorithm for estimating the gradient (5.1) is to form and store an estimate of the second expectation, the vector $\mathbb{E}[\delta\phi]$, and to sample the first expectation, $\mathbb{E}\big[\phi(\phi - \gamma\phi')^\top\big]$. Let u_k denote the estimate of $\mathbb{E}[\delta\phi]$ after observing the first k samples, with $u_0 = 0$. The GTD algorithm is defined by

$$\theta_{k+1} = \theta_k + \alpha_k(\phi_k - \gamma\phi'_k)\phi_k^\top u_k, \tag{5.2a}$$

$$u_{k+1} = u_k + \beta_k(\delta_k\phi_k - u_k), \tag{5.2b}$$

where θ_0 is arbitrary, δ_k is the TD error using θ_k, and $(\alpha_k, \beta_k)_{k\geq 0}$ are sequences of positive step-size parameters, possibly decreasing over time. Notice that if the product is formed right-to-left, then the entire computation is $O(d)$ per-time-step.

GTD, however, is a slow algorithm compared to TD(0). In other words it is poorly conditioned. Let us consider this scenario: assume we can compute u exactly, that is,

$$\begin{aligned} u(\theta) &= \mathbb{E}[\delta(\theta)\phi] \\ &= -A\theta + b, \end{aligned}$$

where A and b are defined in Equation (2.13). Thus, by plugging the exact value of $u(\theta)$ into the θ update, in expectation, the GTD update is driven by matrix $A^\top A$. To see this better, note, from Equation (5.1) we get:

$$-\frac{1}{2}\nabla\text{NEU}(\theta) = \mathbb{E}\Big[\phi - \gamma\phi')\phi^\top\Big]\,\mathbb{E}[\delta(\theta)\phi] \tag{5.3}$$

$$= A^\top(-A\theta + b). \tag{5.4}$$

From concepts of numerical analysis, the condition number of $A^\top A$ is always worse than A—notice $-A$ is the underlying matrix for the expected TD(0) update. As such, GTD's asymptotic rate of convergence is usually much worse than TD(0) on problems where TD(0) converges.

In the next section, we develop two new algorithms, GTD2 and TDC, based on the mean-square projected Bellman-error objective function, which empirically are faster than the GTD algorithm.

5.2 Derivation of the GTD2 and TDC algorithms

In this section we derive two new algorithms, which use mean-square projected Bellman error as their objective function (see Eq. 3.3). We first establish some relationships between the vector-matrix quantities and the relevant statistical expectation terms:

$$\mathbb{E}[\phi\phi^\top] = \sum_s \mu(s)\phi(s)\phi(s)^\top = \Phi^\top D\Phi,$$

where D is a diagonal matrix whose s diagonal element is $\mu(s)$,

$$\begin{aligned}\mathbb{E}[\delta(\theta)\phi] &= \sum_s \mu(s)\phi(s)\left(R(s) + \gamma\sum_{s'} P_{ss'}V_\theta(s') - V_\theta(s)\right)\\ &= \Phi^\top D(TV_\theta - V_\theta),\end{aligned}$$

and note that

$$\begin{aligned}\Pi^\top D\Pi &= (\Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D)^\top D(\Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D)\\ &= D^\top\Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D\Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D\\ &= D^\top\Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D.\end{aligned}$$

Using these relationships, the projected objective can be written in terms of expectations as

$$\begin{aligned}&J(\theta)\\ &= \| V_\theta - \Pi TV_\theta \|_\mu^2\\ &= \| \Pi(V_\theta - TV_\theta) \|_\mu^2\\ &= (\Pi(V_\theta - TV_\theta))^\top D(\Pi(V_\theta - TV_\theta))\\ &= (V_\theta - TV_\theta)^\top\Pi^\top D\Pi(V_\theta - TV_\theta)\\ &= (V_\theta - TV_\theta)^\top D^\top\Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D(V_\theta - TV_\theta)\\ &= (\Phi^\top D(TV_\theta - V_\theta))^\top(\Phi^\top D\Phi)^{-1}\Phi^\top D(TV_\theta - V_\theta)\\ &= \mathbb{E}[\delta(\theta)\phi]^\top\,\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta(\theta)\phi]\,. \qquad (5.5)\end{aligned}$$

Just like in the previous section, which we used a second modifiable parameter $u \in \mathbb{R}^d$ to form a quasi-stationary estimate of all but one of the expectations in the gradient of the NEU objective function (thereby avoiding the need for two independent samples), here, we will use a modifiable parameter $w \in \mathbb{R}^d$, which also involves computing the inverse matrix. Specifically, we use a conventional linear predictor which causes w to estimate

$$w(\theta) = \mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta(\theta)\phi]\,, \qquad (5.6)$$

which looks the same as the solution we get from the LMS method in supervised learning by replacing δ with supervised signals (note $w(\theta) = \mathbb{E}[\phi\phi^\top]^{-1}u(\theta)$). Using this, we can write the negative gradient of the MSPBE objective function as

$$\begin{aligned} -\frac{1}{2}\nabla J(\theta) &= \mathbb{E}\Big[(\phi - \gamma\phi')\phi^\top\Big]\,\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta(\theta)\phi] \\ &= \mathbb{E}\Big[(\phi - \gamma\phi')\phi^\top\Big]\,w(\theta), \end{aligned} \tag{5.7}$$

which can be directly sampled. The resultant $O(d)$ algorithm is

$$\theta_{k+1} = \theta_k + \alpha_k(\phi_k - \gamma\phi'_k)(\phi_k^\top w_k), \tag{5.8a}$$

$$w_{k+1} = w_k + \beta_k(\delta_k - \phi_k^\top w_k)\phi_k, \tag{5.8b}$$

which we call GTD2 algorithm. Note, there is no inverse matrix in the (5.8b) update. We can see that by fixing θ_k to θ, the w-update leads to LMS solution, that is, $w(\theta) = \mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta(\theta)\phi]$.

The derivation of our main algorithm, TDC, starts from the same expression for the gradient and then takes a slightly different route. That is,

$$\begin{aligned} &-\frac{1}{2}\nabla J(\theta) \\ &= \mathbb{E}\Big[(\phi - \gamma\phi')\phi^\top\Big]\,\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta(\theta)\phi] \\ &= \Big(\mathbb{E}\Big[\phi\phi^\top\Big] - \gamma\mathbb{E}\Big[\phi'\phi^\top\Big]\Big)\,\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta(\theta)\phi] \\ &= \mathbb{E}[\delta\phi] - \gamma\mathbb{E}\Big[\phi'\phi^\top\Big]\,\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta(\theta)\phi] \\ &= \mathbb{E}[\delta\phi] - \gamma\mathbb{E}\Big[\phi'\phi^\top\Big]\,w(\theta), \end{aligned} \tag{5.9}$$

which is then sampled, resulting in the following $O(d)$ algorithm, which we call linear TD with gradient correction term, or linear TDC for short:

$$\theta_{k+1} = \theta_k + \alpha_k\left[\delta_k\phi_k - \gamma\phi'_k(\phi_k^\top w_k)\right], \tag{5.10}$$

where w_k is generated by (5.8b) as in GTD2. Note that the update to θ_k is the sum of two terms, and that the first term is exactly the same as the update (2.10) of conventional linear TD. The second term is essentially an adjustment or correction of the TD update so that it follows the gradient of the MSPBE objective function. If the second parameter vector is initialized to $w_0 = 0$, and β_k is small, then this algorithm will start out making almost the same updates as conventional linear TD.

The TDC algorithm (5.10), is derived and works on a given bag of sub-samples—in the form of triples (S_k, R_k, S'_k) that match with both the behavior and target policy sample transitions. What if we wanted to use all the data? Notice that data is generated according to the behavior policy π_b, while our objective is to learn about target policy π. For details, see Section 4.2.

Derivation of Linear TDC for importance-sampling scenario: The objective function that we would like to minimize is $J(\theta) = \| V_\theta - \Pi T^\pi V_\theta \|^2_\mu$, however, because the data is generated according to behavior policy π_b we use importance-sampling. Using Lemma 1 (in Chapter 4), and Equation (4.2) we get:

$$\begin{aligned} J(\theta) &= \|V_\theta - \Pi T^\pi V_\theta\|^2_\mu \\ &= (\mathcal{P}^\pi_\mu \delta(\theta)\phi)^\top \mathbb{E}[\phi_t \phi_t]^{-1} (\mathcal{P}^\pi_\mu \delta(\theta)\phi), \\ &= \mathbb{E}[\rho_t \delta_t(\theta)\phi_t]^\top \mathbb{E}[\phi_t \phi_t]^{-1} \mathbb{E}[\rho_t \delta_t(\theta)\phi_t] \,. \end{aligned} \tag{5.11}$$

Following the linear TDC derivation, we get the following algorithm (linear TDC algorithm based on importance weighting scenario) :

$$\theta_{t+1} = \theta_t + \alpha_t \rho_t \left[\delta_t \phi_t - \gamma \phi_{t+1}(\phi_t^\top w_t)\right], \tag{5.12a}$$

$$w_{t+1} = w_t + \beta_t(\rho_t \delta_t - \phi_t^\top w_t)\phi_t. \tag{5.12b}$$

Similarly, importance weighted version of GTD2, will follow as well.

In the next section, we provide convergence analysis for the linear GTD, GTD2 and TDC algorithms based on sub-sampling and i.i.d. formulation of off-policy learning.

5.3 Convergence Analysis

The purpose of this section is to prove that our new TD algorithms converge to the TD-solution with probability (for the case of off-policy i.i.d. formulation). All of our algorithms are in the class of recursive stochastic algorithms; that is, they are in the form of

$$x_{k+1} = x_k + \alpha_k \left(h(x_k) + M_{k+1}\right), \tag{5.13}$$

where $x \in \mathbb{R}^d$, $h : \mathbb{R}^d \to \mathbb{R}^d$ is a differentiable function, $(\alpha_k)_{k\geq 0}$ denotes positive step-size sequence and $(M_k)_{k\geq 0}$ is a noise sequence.

For the convergence proof, we use the ordinary-differential-equation (ODE) approach. This is a powerful method that allows us to analyze recursive stochastic algorithms by studying their corresponding ODE equation; in short, it says that equation (5.13) is a noisy discretization of

$$\dot{x}(t) = h(x(t)),$$

where $\dot{x}$ denotes derivative of x w.r.t time t; that is $\dot{x} = dx/dt$. Thus, we can study the algorithm through the convergence behavior of the ODE. To use the ODE approach, recursive stochastic algorithms must satisfy certain conditions in Theorem 2.2 of Borkar & Meyn (2000). Before mentioning what this theorem is about, first, we state the required conditions for the purpose of our proofs.

For the iterate (5.13), consider the following conditions:

(C1) $\alpha_k, \; \forall k$, and are deterministic.

(C2) $\sum_k \alpha_k = +\infty$, $\sum_k \alpha_k^2 < \infty$.

(C3) The function h is Lipschitz and $h_\infty(x) = \lim_{c \to \infty} h(cx)/c$ is well-defined for every $x \in \mathbb{R}^d$.

(C4) The sequence $(M_k)_{k \geq 0}$ is a martingale difference sequence w.r.t. increasing family of σ-fields, $\mathcal{F}_k \overset{\text{def}}{=} \sigma(x_0, M_1, \ldots, M_k)$; that is, $\mathbb{E}[M_{k+1} \,|\, \mathcal{F}_k] = 0$.

(C5) For some constant $K > 0$, $\mathbb{E}\big[\|M_{k+1}\|^2 \,|\, \mathcal{F}_k\big] \leq K(1 + \|x_k\|^2)$, holds for any $k \geq 0$ almost surely.

(C6) The ODE, $\dot{x} = h_\infty(x)$, has the origin as its unique globally asymptotically stable equilibrium.

The analysis plan is as follows: first we provide the ODE Lemma, which is a pre-requisite for our proof of convergence.

Lemma 2. *(The ODE Lemma) Consider the iteration (5.13), and assume that the conditions (C1)-(C6) holds. Then, as $k \to \infty$, the x_k converges with probability one to the unique globally asymptotically stable equilibrium of the ODE, $\dot{x}(t) = h(x(t))$.*

Proof. For the proof see the theorem 2.2 of Borkar and Meyn (2000). □

5.3.1 Convergence analysis for GTD

Here, we provide convergence analysis for GTD. First, let us consider the following assumptions:

(A1) $S_k \sim \mu(\cdot)$, $R_k = R(S_k, A_k, S'_k)$ and $(R_k, S'_k) \sim P(\cdot, \cdot \mid S_k, A_k)$.

(A2) $(S_k, R_k, S'_k)_{\geq 0}$ is an i.i.d sequence.

(A3) $(\phi_k, \phi'_k)_{k\geq 0}$ is uniformly bounded second moments (unconditional), where $\phi_k = \phi(S_k), \phi'_k = \phi(S'_k)$.

(A4) $\exists \hat{R}_{\max}$ s.t. Var $[R_k|S_k] \leq \hat{R}_{\max}$ holds almost surely (a.s.).

Consider the following (standard) assumptions on features:

(P1) $\|\phi_k\|_\infty < +\infty, \ \|\phi'_k\|_\infty < +\infty, \quad \forall k.$

(P2) The matrices $C = \mathbb{E}\big[\phi_k \phi_k^\top\big]$ and $A = \mathbb{E}\big[\phi_k(\phi_k - \gamma\phi'_k)^\top\big]$ are non-singular and uniformly bounded. Note that A and b are well-defined because the distribution of (ϕ_k, R_k, ϕ_k) does not depend on the sequence index k.

And, the following step-size conditions:

(S1) $\alpha_k, \beta_k > 0, \forall k$, and are deterministic.

(S2) $\sum_k \alpha_k = \sum_k \beta_k = +\infty$, $\sum_k \alpha_k^2 < \infty$, and $\sum_k \beta_k^2 < \infty$.

Theorem 1 (Convergence of GTD). *Consider the GTD iterations (5.2a) and (5.2b) with step-size conditions (S1)-(S2), and let the step-size sequences α_k and β_k satisfy $\beta_k = \eta\alpha_k$, $\eta > 0$. Assume that (A1)-(A4), (P1)-(P2) holds. Then the parameter vector θ_k converges to the TD-solution with probability one.*

Proof. We use Lemma 2 for our proof of convergence.

We rewrite the GTD algorithm's two iterations (eqs. (5.2a)-(5.2b)) as a single iteration in a combined parameter vector with $2d$ components, $\varrho_k^\top = (v_k^\top, \theta_k^\top)$, where $v_k = u_k/\sqrt{\eta}$. We also define $g_{k+1}^\top = (R_k\phi_k^\top, 0^\top)$— a new reward-related vector with $2d$ components. Thus,

$$\varrho_{k+1} = \varrho_k + \alpha_k\sqrt{\eta}\left(G_{k+1}\varrho_k + g_{k+1}\right), \tag{5.14}$$

where

$$G_{k+1} = \left(\begin{array}{cc} -\sqrt{\eta} I & \phi_k(\gamma {\phi'}_k - \phi_k)^\top \\ -(\gamma {\phi'}_k - \phi_k)\phi_k^\top & 0 \end{array} \right).$$

Let $G = \mathbb{E}[G_k]$ and $g = \mathbb{E}[g_k]$. Note that G and g are well-defined as by the assumption the process $(\phi_k, R_k, \phi'_k)_{k\geq 0}$ is i.i.d. In particular,

$$G = \left(\begin{array}{cc} -\sqrt{\eta} I & -A \\ A^\top & 0 \end{array} \right), \quad g = \left(\begin{array}{c} b \\ 0 \end{array} \right).$$

Further, note that the TD solution follows from $G\varrho + g = 0$, where the w-fixpoint is zero.

Now, we re-write the above iterate in the following form:

$$\begin{aligned} \varrho_{k+1} &= \varrho_k + \alpha_k \sqrt{\eta}(G\varrho_k + g + (G_{k+1} - G)\varrho_k + (g_{k+1} - g)) \\ &= \varrho_k + \alpha'_k (h(\varrho_k) + M_{k+1}), \end{aligned}$$

where $\alpha'_k = \alpha_k \sqrt{\eta}$, $h(\varrho) = g + G\varrho$ and $M_{k+1} = (G_{k+1} - G)\varrho_k + g_{k+1} - g$. Let $\mathcal{F}_k = \sigma(\varrho_1, M_1, \ldots, \varrho_{k-1}, M_k)$ be σ-fields generated by the quantities ϱ_i, M_i, $i \leq k$, $k \geq 1$. Lemma 2 requires the verification of the following conditions:

(i) The function h is Lipschitz and $h_\infty(\varrho) = \lim_{c\to\infty} h(c\varrho)/c$ is well-defined for every $\varrho \in \mathbb{R}^{2d}$;

(ii-a) The sequence $(M_k, \mathcal{F}_k)$ is a martingale difference sequence, and

(ii-b) for some $c_0 > 0$, $\mathbb{E}\big[\|M_{k+1}\|^2 \,|\, \mathcal{F}_k\big] \leq c_0(1 + \|\varrho_k\|^2)$ holds for any initial parameter vector ϱ_1;

(iii) The sequence α'_k satisfies, $\sum_{k=1}^\infty \alpha'_k = \infty$, $\sum_{k=1}^\infty (\alpha'_k)^2 < +\infty$;

(iv) The ODE $\dot{\varrho} = h_\infty(\varrho)$ has the origin as a globally asymptotically stable equilibrium, and

(v) The ODE $\dot{\varrho} = h(\varrho)$ has a unique globally asymptotically stable equilibrium.

Clearly, $h(\varrho)$ is Lipschitz with coefficient $\|G\|$ and $h_\infty(\varrho) = G\varrho$. By construction, $(M_k, \mathcal{F}_k)$ satisfies $\mathbb{E}[M_{k+1}|\mathcal{F}_k] = 0$ and $M_k \in \mathcal{F}_k$, i.e., it is a martingale difference sequence. Condition (ii-b) can be shown to hold by a simple application of the triangle inequality and the boundedness of the second moments of (ϕ_k, R_k, ϕ'_k). Condition (iii) is satisfied by our conditions on the step-size sequences α_k, β_k.

For the last two conditions, we begin by showing that the real-part of all the eigenvalues of G are negative.

Let $\lambda \in \mathbb{C}$, $\lambda \neq 0$ be an eigenvalue of G with corresponding normalized eigenvector $x \in \mathbb{C}^{2d}$; that is, $\|x\|^2 = x^* x = 1$, where x^* is the complex conjugate of x. Hence $x^* G x = \lambda$. Let $x^\top = (x_1^\top, x_2^\top)$, where $x_1, x_2 \in \mathbb{C}^d$. Using the definition of G, $\lambda = x^* G x = -\sqrt{\eta}\|x_1\|^2 - x_1^* A x_2 + x_2^* A^\top x_1$, where $\|x_1\|^2 = x_1^* x_1$. Because A is real, $A^* = A^\top$, and it follows that $(x_1^* A x_2)^* = x_2^* A^\top x_1$. Thus, $\mathrm{Re}(\lambda) = \mathrm{Re}(x^* G x) = -\sqrt{\eta}\|x_1\|^2 \leq 0$. We are now done if we show that x_1 cannot be zero. If $x_1 = 0$, then from $\lambda = x^* G x$ we get $\lambda = 0$, however, because, G is non-singular, $\lambda \neq 0$. Thus, x_1 can not be zero.

To show G is non-singular, we can use the determinant rule for partitioned matrices. According to this rule, if $A_1 \in \mathbb{R}^{d_1 \times d_1}, A_2 \in \mathbb{R}^{d_1 \times d_2}, A_3 \in \mathbb{R}^{d_2 \times d_1}, A_4 \in \mathbb{R}^{d_2 \times d_2}$ then for $U = [A_1 A_2; A_3 A_4] \in \mathbb{R}^{(d_1+d_2)\times(d_1+d_2)}, \det(U) = \det(A_1)\det(A_4 - A_3 A_1^{-1} A_2)$. Using this rule, we get $\det(G) = \det(A^\top A) = (\det A)^2 \neq 0$. This indicates that all the eigenvalues of G are non-zero.

Finally, for the ODE $\dot{\varrho} = h(\varrho)$, note that $\varrho^* = -G^{-1} g$ is the unique asymptotically stable equilibrium with $\bar{V}(\varrho) = (G\varrho + g)^T (G\varrho + g)/2$ as its associated strict Lyapunov function. Also note, $h_\infty(\varrho) = \lim_{c \to \infty} \dfrac{h(c\varrho)}{c}$. Then $h_\infty(\varrho) = G\varrho$. For the ODE

$$\dot{\varrho} = h_\infty(\varrho) = G\varrho,$$

the origin is a globally asymptotically stable equilibrium because all the real-part eigenvalues of the G matrix are negative.

The claim now follows. □

5.3.2 Convergence analysis for GTD2

Theorem 2 (Convergence of GTD2)**.** *Consider the GTD2 iterations (5.8a) and (5.8b). Assume that (A1)-(A4), (P1)-(P2), (S1)-(S2), holds, and let the step-size sequences α_k and β_k satisfy $\beta_k = \eta\alpha_k$, $\eta > 0$. Then then parameter vector θ_k converges to the TD-solution and u_k converges to zero with probability one.*

Proof. The proof is very similar to the convergence proof for GTD. Thus, we omit many overlapping parts. First, we rewrite GTD2's two iterations (eqs. (5.8a) and (5.8b)) as a single iteration in a combined parameter vector with $2d$ components, $\varrho_k^\top = (v_k^\top, \theta_k^\top)$, where $v_k = w_k/\sqrt{\eta}$, and a new reward-related vector with $2d$ components, $g_{k+1}^\top = (\eta r_k \phi_k^\top, 0^\top)$, as follows:

$$\varrho_{k+1} = \varrho_k + \alpha_k \left(G_{k+1}\varrho_k + g_{k+1} \right), \tag{5.15}$$

where

$$G_{k+1} \;=\; \left(\begin{array}{cc} -\sqrt{\eta} C & \phi_k(\gamma\phi'_k - \phi_k)^\top \\ -(\gamma\phi'_k - \phi_k)\phi_k^\top & 0 \end{array} \right).$$

Let $G = \mathbb{E}[G_k]$ and $g = \mathbb{E}[g_k]$. Note that G and g are well-defined as by the assumption the process $(\phi_k, R_k, \phi'_k)_{k\geq 0}$ is stationary. In particular,

$$G = \left(\begin{array}{cc} -\sqrt{\eta} C & -A \\ A^\top & 0 \end{array} \right), \quad g = \left(\begin{array}{c} b \\ 0 \end{array} \right).$$

Further, note that TD solution follows from $G\varrho + g = 0$, where w-fixpoint is zero.

The proof is similar to the convergence proof of GTD, which required verification of conditions in Lemma 2. Thus, similarly, we can verify all the conditions. Here, we only need to show that all real-part eigenvalues of G are negative. First, we show that G is non-singular. Using the determinant rule for partitioned matrices, we get $\det(G) = \det(A^\top A) = (\det A)^2 \neq 0$. This indicates that all the eigenvalues of G are non-zero.

Now, let $\lambda \in \mathbb{C}$, $\lambda \neq 0$ be an eigenvalue of G with corresponding normalized eigenvector $x \in \mathbb{C}^{2n}$; that is, $\|x\|^2 = x^*x = 1$, where x^* is the complex conjugate of x. Hence $x^*Gx = \lambda$. Let $x^\top = (x_1^\top, x_2^\top)$, where $x_1, x_2 \in \mathbb{C}^n$. Using the definition of G, $\lambda = x^*Gx = -\sqrt{\eta}\|x_1\|_C^2 - x_1^*Ax_2 + x_2^*A^\top x_1$, where $\|x_1\|_C^2 = x_1^*Cx_1$. Because A is real, $A^* = A^\top$, and it follows that $(x_1^*Ax_2)^* = x_2^*A^\top x_1$. Thus, $\mathrm{Re}(\lambda) = \mathrm{Re}(x^*Gx) = -\sqrt{\eta}\|x_1\|_C^2 \leq 0$. We are now done if we show that x_1 cannot be zero. If $x_1 = 0$, then from $\lambda = x^*Gx$ we get that $\lambda = 0$, which contradicts with $\lambda \neq 0$. □

5.3.3 Convergence analysis for TDC

Theorem 3 (Convergence of TDC)**.** *Consider the TDC iterations (5.10) and (5.8b), and $w_0 = 0$. Assume that (A1)-(A4), (P1)-(P2), (S1)-(S2), holds, and let the step-size sequences α_k and β_k satisfy $\beta_k = \eta\alpha_k$, $\eta > 0$. Consider the matrix A and C in (P2). Let*

$$H(A) \overset{\text{def}}{=} \frac{(A + A^\top)}{2},$$

and $\lambda_{\min}\left(C^{-1}H(A)\right)$ be the minimum eigenvalue of the matrix $C^{-1}H(A)$. If

$$\eta > \max\{0, -\lambda_{\min}\left(C^{-1}H(A)\right)\},$$

then the learning parameter θ_k converges to the TD solution with probability one.

Proof. The proof is similar to the convergence proof of GTD/GTD2, which requires verification of conditions in Lemma 2. The only subtle difference is verification of the condition on η.

First, we rewrite the TDC algorithm's two iterations (eqs. 5.10 and 5.8b) as a single iteration in a combined parameter vector with $2d$ components, $\varrho_k^\top = (w_k^\top, \theta_k^\top)$, and a new reward-related vector with $2d$ components, $g_{k+1}^\top = (\eta R_k \phi_k^\top, R_k \phi_k^\top)$, as follows:

$$\varrho_{k+1} = \varrho_k + \alpha_k \left(G_{k+1} \varrho_k + g_{k+1} \right), \tag{5.16}$$

where

$$G_{k+1} = \begin{pmatrix} -\eta\phi_k\phi_k^\top & \eta\phi_k(\gamma{\phi'}_k - \phi_k)^\top \\ -\gamma\phi'_k\phi_k^\top & \phi_k(\gamma{\phi'}_k - \phi_k)^\top \end{pmatrix}.$$

Let $G = \mathbb{E}[G_k]$ and $g = \mathbb{E}[g_k]$ and $F = \gamma\mathbb{E}\left[\phi_k {\phi'}_k^\top\right]$, thus, it can be seen $A = C - F$. Note that G and g are well-defined as by the assumption the process $(\phi_k, R_k, \phi'_k)_{k\geq 0}$ is i.i.d. In particular,

$$G = \begin{pmatrix} -\eta C & -\eta A \\ -F^\top & -A \end{pmatrix}, \quad g = \begin{pmatrix} \eta b \\ b \end{pmatrix}.$$

Also, note that the TD solution follows from $G\varrho + g = 0$, where the w-fixpoint is zero.

Because the linear system is similar to what we obtained for the GTD/GTD2 proof of convergence. Here, we only need to verify that all the real-part eigenvalues of matrix G are negative.

The eigenvalue of matrix G, $\lambda \in \mathbb{C}$, satisfies in the polynomial equation

$$\begin{aligned} &\det(G - \lambda I) \\ &\quad = \det\begin{pmatrix} -\eta C - \lambda I & -\eta A \\ -F^\top & -A - \lambda I \end{pmatrix} \\ &\quad = \det\begin{pmatrix} \eta C_{\lambda,\eta} & \eta A \\ C - A^\top & A + \lambda I \end{pmatrix} = 0, \end{aligned}$$

where we used $\det(-X) = (-1)^{2d}\det(X) = \det(X)$, $X \in \mathbb{R}^{2d\times 2d}$, and where

$$C_{\lambda,\eta} = C + \frac{\lambda}{\eta} I.$$

Let us assume that $C_{\lambda,\eta}$ is non-singular, then

$$\begin{aligned}\det(G-\lambda I) &= \det(\eta C_{\lambda,\eta})\,\det\Big(A+\lambda I-(C-A^\top)\,(\eta C_{\lambda,\eta})^{-1}\,\eta A\Big)\\ &= \eta^{2d}\det(C_{\lambda,\eta})\,\det\Big(A+\lambda I-(C-A^\top)\,C_{\lambda,\eta}^{-1}\,A\Big),\end{aligned}$$

where we used the determinant rule for block matrices (see GTD's proof of convergence in Theorem 1).

Using $X(Y+X)^{-1} = (Y+X-Y)(Y+X)^{-1} = I - Y(Y+X)^{-1}$, we get that $CC_{\lambda,\eta}^{-1} = I - \frac{\lambda}{\eta}C_{\lambda,\eta}^{-1}$. Thus,

$$\begin{aligned}A+\lambda I-(C-A^\top)C_{\lambda,\eta}^{-1}A &= A+\lambda I - CC_{\lambda,\eta}^{-1}A + A^\top C_{\lambda,\eta}^{-1}A\\ &= \lambda I + \frac{\lambda}{\eta}C_{\lambda,\eta}^{-1}A + A^\top C_{\lambda,\eta}^{-1}A\\ &= (\lambda A^{-1}C_{\lambda,\eta} + A^\top + \frac{\lambda}{\eta}I)\,C_{\lambda,\eta}^{-1}A\\ &= A^{-1}(\lambda C_{\lambda,\eta} + A(A^\top + \frac{\lambda}{\eta}I))\,C_{\lambda,\eta}^{-1}A.\end{aligned}$$

Hence,

$$\det\begin{pmatrix} -\eta C-\lambda I & -\eta A\\ -F^\top & -A-\lambda I\end{pmatrix} = \eta^{2d}\,\det(\lambda C_{\lambda,\eta} + A(A^\top + \frac{\lambda}{\eta}I)).$$

Because an eigenvalue of the matrix G, λ, satisfies $\det(G-\lambda I) = 0$, there must exist a nonzero vector $x \in \mathbb{C}^d$, such that

$$x^*\left(\lambda C_{\lambda,\eta} + A(A^\top + \frac{\lambda}{\eta}I)\right)x = 0,$$

where x^* is the complex conjugate of vector x (including its transpose), and $x^*x = \|x\|^2 > 0$. From the above equation, it is easy to get the following quadratic equation in terms of λ,

$$\|x\|^2\lambda^2 + (\eta x^*Cx + x^*Ax)\lambda + \eta\|Ax\|^2 = 0,$$

where $\|Ax\|^2 = x^*A^\top Ax = x^*AA^\top x$. This quadratic equation has two solutions, λ_1 and

λ_2, where [2]

$$\lambda_1\lambda_2 = \frac{\eta\|Ax\|^2}{\|x\|^2},$$

and

$$\lambda_1 + \lambda_2 = \frac{-(\eta x^* Cx + x^* Ax)}{\|x\|^2}.$$

Because $\lambda_1\lambda_2$ is a positive and real number, thus, $\lambda_2 = \kappa\lambda_1^*$, where $\kappa > 0$. Thus, we have $\lambda_1 + \lambda_2 = \lambda_1 + \kappa\lambda_1^*$. Let $\mathrm{Re}(\lambda)$ denote the real-part of λ. Thus, $\mathrm{Re}(\lambda_1 + \lambda_2) = (1+\kappa)\mathrm{Re}(\lambda_1) = (1 + 1/\kappa)\mathrm{Re}(\lambda_2)$. We are done if we show $\mathrm{Re}(\lambda_1 + \lambda_2) < 0$, because that leads to $\mathrm{Re}(\lambda_1) < 0$ and $\mathrm{Re}(\lambda_2) < 0$. First, let us start from

$$\begin{aligned}
\mathrm{Re}(\lambda_1 + \lambda_2) &= \frac{\mathrm{Re}\left(-(\eta x^* Cx + x^* Ax)\right)}{\|x\|^2} \\
&= \frac{-(\eta x^* Cx + x^* Ax) - (\eta x^* Cx + x^* Ax)^*}{2\|x\|^2} \\
&= \frac{-\left(2\eta x^* Cx + x^*(A + A^\top)x\right)}{2\|x\|^2}.
\end{aligned}$$

The condition $\mathrm{Re}(\lambda_1 + \lambda_2) < 0$ is equivalent to

$$2\eta x^* Cx + x^*(A + A^\top)x > 0.$$

Clearly, the above inequality, holds if

$$\eta > \max_{z\neq 0,\ z\in\mathbb{R}^d} \frac{-z^\top H(A)z}{z^\top Cz}, \quad H(A) \overset{\text{def}}{=} \frac{A + A^\top}{2}.$$

Here, the $H(A)$ and C matrices are symmetric, hence their eigenvalues and eigenvectors are reals. For any $z \neq 0$, $z \in \mathbb{R}^d$, let $y = C^{1/2}z$. Then when $z^\top Cz = 1$, we also have $\|y\|^2 = 1$. Therefore, it suffices to have

$$\eta > \max_{\|y\|^2=1} y^\top \left(-C^{-1/2}H(A)C^{-1/2}\right) y,$$

which is equivalent to $\eta > -\lambda_{\min}\left(C^{-1/2}H(A)C^{-1/2}\right)$. Note, $C^{-1/2}H(A)C^{-1/2}$ is symmetric, so its eigenvalues are all real. Also, because

$$\lambda_{\min}\left(C^{-1/2}H(A)C^{-1/2}\right) = \lambda_{\min}\left(C^{-1}H(A)\right),$$

[2]The quadratic equation, $ax^2 + bx + c = 0$ (with $a \neq 0$), has two solutions. The product and the sum of these two solutions are, $x_1x_2 = \frac{c}{a}$ and $x_1 + x_2 = \frac{-b}{a}$, respectively.

then, given $\eta > 0$, we conclude

$$\eta > \max\{0, -\lambda_{\min}\left(C^{-1}H(A)\right)\},$$

which suffices to guarantee the stability of matrix G, thus finishing the proof □

5.3.4 Convergence remarks for TDC with importance-weighting scenario

It can be shown that the TDC's iterates (5.12a) and (5.12b) converge with probability one to the TD solution and zero, respectively, under the standard assumptions in this chapter and Lemma 6.7 (Bertsekas and Tsitsiklis 1996) and conditions therein. For a through proof, we need to substitute the Martingale noise with a Markov noise and then use Theorem 17 in page 239 of Benveniste et al. (1990) (also see Delyon, 1996). The proof is similar to our proofs of convergence in this chapter.

5.4 Empirical results

To begin to assess the practical utility of the new family of TD algorithms based on gradient-descent, including TDC, GTD2 and GTD, we compared their empirical learning rate to that of conventional TD(0) with linear function approximation on four small problems—three random-walk problems and a Boyan-chain problem. All of these problems were episodic, undiscounted, and involved only on-policy training with a fixed policy.

The random-walk problems were all based on the standard Markov chain (Sutton and Barto, 1998) with a linear arrangement of five states plus two absorbing terminal states at each end. Episodes began in the center state of the five, then transitioned randomly with equal probability to a neighbouring state until a terminal state was reached. The rewards were zero everywhere except on transition into the right terminal state, upon which the reward was $+1$.

We used three versions of this problem, differing only in their feature representations. The first representation, which we call *tabular features*, was the familiar table-lookup case in which, for example, the second state was represented by the vector $\phi_2 = (0, 1, 0, 0, 0)^\top$. The second representation, which we call *inverted features*, was chosen to cause extensive inappropriate generalization between states; it represented the second state by $\phi_2 = (\frac{1}{2}, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})^\top$ (the value $\frac{1}{2}$ was chosen to give the feature vectors unit norm). The third representation, which we called *dependent features*, used only $d = 3$ features and was not sufficient to solve the problem exactly. The feature vectors for the five states, left to right, were $\phi_1 = (1, 0, 0)^\top$, $\phi_2 = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0)^\top$, $\phi_3 = (\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})^\top$, $\phi_4 = (0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})^\top$, and $\phi_5 = (0, 0, 1)^\top$.

The Boyan-chain problem is a standard episodic task for comparing TD-style algorithms with linear function approximation (see Boyan (2002) for details). We used the version with 14 states and $d = 4$ features.

We applied GTD, GTD2, TDC, and TD(0) to these problems with a range of constant values for their step-size parameters. The parameter α was varied over a wide range of values, in powers of 2. For the GTD, GTD2, and TDC algorithms, the ratio $\eta = \beta/\alpha$ took values from the set $\{\frac{1}{4}, \frac{1}{2}, 1, 2\}$ for the random-walk problems; one lower power of two was added for the Boyan-chain problem. The initial parameter vectors, θ_0 and w_0, were set to 0 for all algorithms.

Each algorithm and parameter setting was run for 100-500 episodes depending on the problem, with the square root of the MSPBE, MSBE, NEU, and MSE (see Section 3) computed after each episode, then averaged over 100 independent runs. Figure 5.1 summarizes all the results on the small problems using the MSPBE as the performance measure[3]. The results for the other objective functions were similar in all cases and produced the same rankings. The standard errors are all very small (in the order of 10^{-2} to 10^{-3}), thus, are not shown. All the algorithms were similar in terms of their dependence and sensitivity to the step sizes. Overall, GTD learned the slowest, followed after a significant margin by GTD2, followed by TDC and TD(0). It is important to note that our step-sizes in these empirical results are kept constant and as a result the RMSPBE as shown in Figure 5.1 (right sub-panel) will never go to zero.

To get a measure of how well the new algorithms perform on a larger problem, David Silver applied them to learning an evaluation function for 9x9 Computer Go. He used a version of RLGO (Silver et al. 2007) modified to use a purely linear evaluation function. This system used 969,894 binary features corresponding to all possible shapes in every 3x3, 2x2, and 1x1 region of the board. Using weight sharing to take advantage of location-independent and location-dependent symmetries, the million features are reduced to a parameter vector of $n =$ 63,303 components. With this large of a parameter vector, $O(d^2)$ methods are not feasible. To make the problem as straightforward as possible, he sought to learn the value function for a fixed policy, in this case for the policy that chose randomly among the legal moves. Experience was generated by self-play, with all rewards zero except upon winning the game, when the reward was 1.

[3]In this thesis, I have corrected the results of Figure 3 in Sutton et al. (2009). It turns out some of those graphs have mislabeling issues, which have been fixed here. Also note, the general conclusions still remain the same.

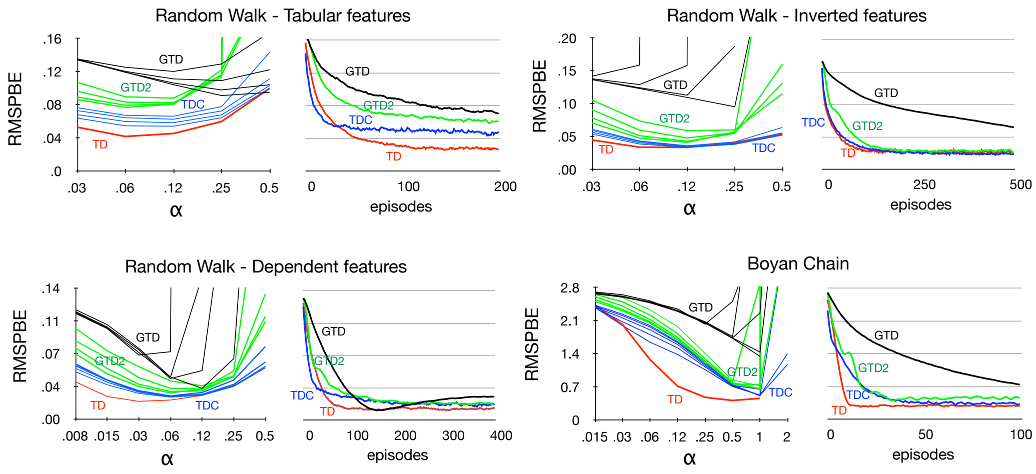

Figure 5.1: Empirical results on the four small problems—three versions of the 5-state random walk plus the 14-state Boyan chain. In each of the four panels, the right sub-panel shows a learning curve at best parameter values (RMSPBE denotes root of MSPBE objective function), and the left sub-panel shows a parameter study plotting the average height of the learning curve for each algorithm, for various $\eta = \beta/\alpha$, as a function of α. TD label shown in the graph represents TD(0) algorithm.

He applied all four algorithms to this problem with a range of step sizes. In each run, θ was initialized to random values uniformly distributed in $[-0.1, 0.1]$. The secondary parameter, w, was initialized to 0. Training then proceeded for 1000 complete games, after which θ was frozen and another 1000 games run to compute an estimate of an objective function. The objective functions cannot be exactly computed here because of the size of the problem.

The NEU objective is the most straightforward to estimate simply by averaging the value of $\delta\phi$ over the 1000 test games and then taking the norm of the resultant vector. It is this performance measure that he recorded and averaged over 40 runs to produce the data shown in Figure 5.2 (Sutton, Maei, et al., 2009).

The results are remarkably consistent with what we saw in the small problems. The GTD algorithm was the slowest, followed by GTD2, TDC, and TD, though the differences between the last three are probably not significant given the coarseness of the parameter sampling and the standard errors, which were about 0.05 in this experiment (they are omitted from the graph to reduce clutter).

Finally, Figure 5.3 shows the results for an off-policy learning problem, demonstrating that the gradient methods converge on Baird's counterexample for which TD diverges.

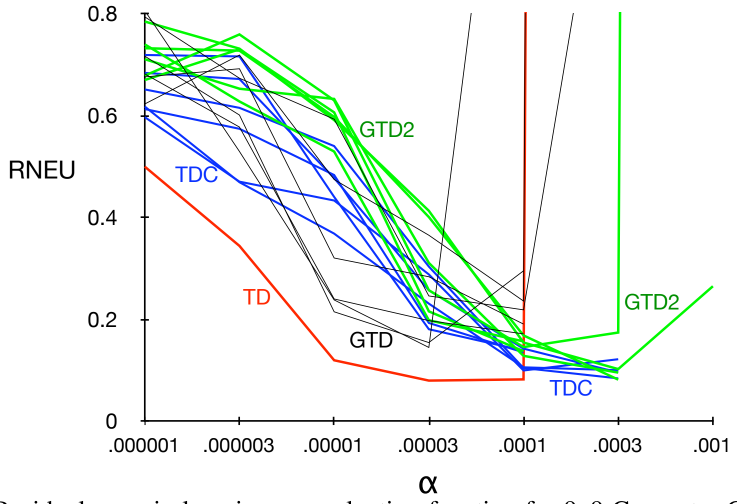

Figure 5.2: Residual error in learning an evaluation function for 9x9 Computer Go, as a function of algorithm and step-size, in terms of the square root of the norm of the expected TD update (3.4). The η values used were $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$, 1, and 2.

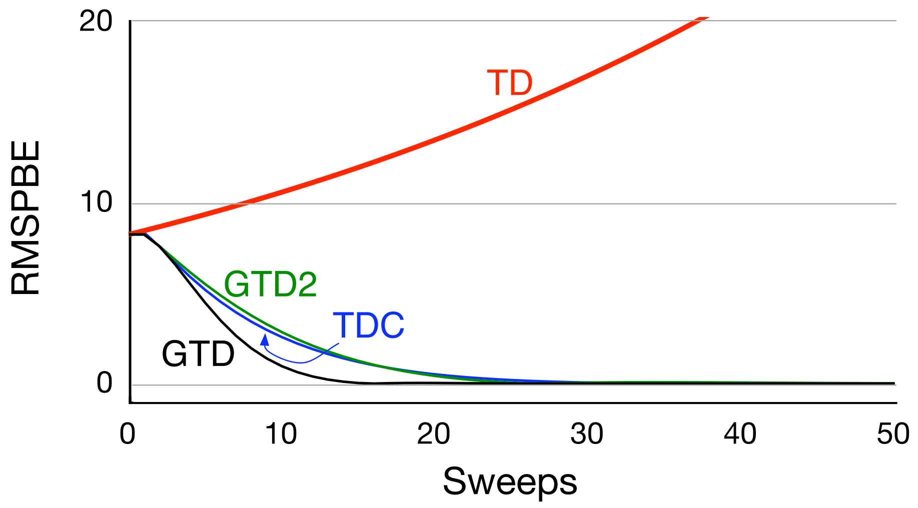

Figure 5.3: Learning curves on Baird's off-policy counterexample: TD diverges, whereas the gradient methods converge. This is the 7-state version of the "star" counterexample (Baird 1995), for which divergence is monotonic. Updating was done synchronously in dynamic-programming-like sweeps through the state space. For TD, $\alpha = 0.1$. For the gradient algorithms, $\alpha = 0.05$ and $\eta = 10$. The initial parameter value was $\theta_0 = (1, 1, 1, 1, 1, 1, 10, 1)^\top$, and $\gamma = 0.99$.

5.5 Conclusion

In this chapter, we introduced three new gradient-based temporal-difference learning algorithms. GTD minimizes the NEU objective function while GTD2 and TDC minimize a more natural performance measure— the mean-square projected Bellman error. We provided a convergence analysis for our algorithms. Among our new algorithms, TDC appears to be more efficient than GTD and GTD2, in terms of empirical rate of convergence. However, TDC appears to be a bit slower than TD(0) on on-policy problems. In the next chapter we develop nonlinear version of GTD2 and TDC.

Chapter 6

Nonlinear Gradient Temporal-Difference Learning

In previous chapter we showed how to solve the problem of off-policy learning with linear TD algorithms by introducing the projected Bellman-error objective function, and algorithms that perform stochastic gradient-descent on this function. These methods can be viewed as natural generalizations of previous TD methods, as they converge to the same limit points when used with linear function approximation methods.

In this chapter [1], we generalize this work to non-linear function approximation. We present a projected Bellman-error objective function and two gradient-descent TD algorithms, non-linear GTD2 and TDC, that optimize it. We prove the asymptotic almost-sure convergence of both algorithms, for any finite Markov decision process and any smooth value function approximator, to a locally optimal solution. The algorithms are incremental and the computational complexity per-time-step scales linearly with the number of parameters of the approximator. Empirical results obtained in the game of Go[2] demonstrate the algorithms effectiveness.

6.1 Nonlinear TD learning

Just like previous chapter, we assume that we are given an infinite sequence of 3-tuples, (S_k, R_k, S'_k), that satisfies the following:

[1]This chapter is based on the following paper: Maei, H. R., Szepesvari, Cs, Bhatnagar, S., Precup, D., Silver D., Sutton, R. S. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems 22*. MIT Press.

[2]All the Computer Go results were obtained by David Silver. The results were published in our paper: Maei, et al. (2009).

Assumption A1 $(S_k)_{k\geq 0}$ is an $\mathcal{S}$-valued[3] stationary process, $S_k \sim \mu(\cdot)$, $R_k = R(S_k)$ and $S'_k \sim P^\pi(\cdot|S_k)$.

We call (S_k, R_k, S'_k) the k^{th} transition, and we consider the samples to have underlying i.i.d. distribution (see Chapter 5). Upon observing the k^{th} transition, we compute the scalar-valued *temporal-difference error*,

$$\delta_k = R_k + \gamma V_{\theta_k}(S'_k) - V_{\theta_k}(S_k),$$

which is then used to update the parameter vector as follows:

$$\theta_{k+1} \leftarrow \theta_k + \alpha_k \, \delta_k \nabla V_{\theta_k}(S_k). \tag{6.1}$$

Here α_k is a deterministic positive step-size parameter, which is typically small, or (for the purpose of convergence analysis) is assumed to satisfy the Robbins-Monro conditions: $\sum_{k=0}^{\infty} \alpha_k = \infty$, $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$. We denote by $\nabla V_\theta(s) \in \mathbb{R}^d$ the gradient of V w.r.t. θ at s.

Because we assume stationarity, we will sometimes drop the index k and use (S, R, S') to denote a random transition. When the TD(0) algorithm converges, it must converge to a parameter value where, in expectation, the parameters do not change:

$$\mathbb{E}[\delta(\theta) \, \nabla V_\theta(S)] = 0, \tag{6.2}$$

where S, δ are random and share the common distribution underlying (S_k, δ_k); in particular, (S, R, S') are drawn as in Assumption A1 and $\delta(\theta) = R + \gamma V_\theta(S') - V_\theta(S)$.

However, it is well known that TD(0) with nonlinear function approximation may not converge (even for on-policy problems); the stability of the algorithm is affected both by the actual function approximation V_θ and by the way in which transitions are sampled.

Our goal is to generalize our gradient-TD method approach, which we described in previous chapter, to the case in which V_θ is a smooth, nonlinear function approximator. The first step is to find a good objective function on which to do gradient descent. In the linear case, MSPBE was chosen as a projection of the Bellman error on a natural hyperplane–the subspace to which V_θ is restricted. However, in the nonlinear case, the value function is no longer restricted to a plane, but can move on a nonlinear surface.

[3]Note, $\mathcal{S}$ is not necessarily finite.

6.2 Objective function for nonlinear function approximation

To find an analogous MSPBE objective function for nonlinear function approximation consider the following scenario: Assume that V_θ is a differentiable function of θ, and thus $\mathcal{M} = \{V_\theta \in \mathbb{R}^{|S|} \,|\, \theta \in \mathbb{R}^d\}$ becomes a differentiable sub-manifold of $\mathbb{R}^{|S|}$. Projecting onto a nonlinear manifold is not computationally feasible; to get around this problem, we will assume that the parameter vector θ changes very little in one step (given that the value function is smooth and learning rates are usually small); in this case, the surface is locally close to linear, and we can project onto the tangent plane at the given point. We now detail this approach and show that this is indeed a good objective function.

The *tangent plane* $P\mathcal{M}_\theta$ of $\mathcal{M}$ at θ is the hyperplane of $\mathbb{R}^{|S|}$ that *(i)* passes through V_θ and *(ii)* is orthogonal to the normal of $\mathcal{M}$ at θ. The *tangent space* $T\mathcal{M}_\theta$ is the translation of $P\mathcal{M}_\theta$ to the origin. Note that $T\mathcal{M}_\theta = \{\Phi_\theta a \,|\, a \in \mathbb{R}^d\}$, where $\Phi_\theta \in \mathbb{R}^{|S|\times d}$ is defined by $(\Phi_\theta)_{s,i} = \frac{\partial}{\partial \theta_i} V_\theta(s)$. Let Π_θ be the projection that projects vectors of $(\mathbb{R}^{|S|}, \|\cdot\|_\mu)$ to $T\mathcal{M}_\theta$. If $\Phi_\theta^\top D \Phi_\theta$ is non-singular then Π_θ can be written as

$$\Pi_\theta = \Phi_\theta(\Phi_\theta^\top D \Phi_\theta)^{-1}\Phi_\theta^\top D, \tag{6.3}$$

where D is a diagonal matrix whose s^{th} diagonal component is $\mu(s)$.

The objective function that we will optimize is:

$$J(\theta) \quad = \quad \| \, \Pi_\theta(T^\pi V_\theta - V_\theta) \, \|_\mu^2, \tag{6.4}$$

where T^π represents Bellman operator, which is defined in Equation (2.2). For simplicity we will drop the superscript π.

This is a natural generalization of the objective function defined by (3.3), as the plane on which we project is parallel to the tangent plane at θ. More precisely, let Υ_θ be the projection to $P\mathcal{M}_\theta$ and let Π_θ be the projection to $T\mathcal{M}_\theta$. Because the two hyperplanes are parallel, for any $V \in \mathbb{R}^{|S|}$, $\Upsilon_\theta V - V_\theta = \Pi_\theta(V - V_\theta)$. In other words, projecting onto the tangent space gives exactly the same distance as projecting onto the tangent plane, while being mathematically more convenient. Fig. 6.1 illustrates visually this objective function.

We now show that $J(\theta)$ can be re-written in the same way as done in Chapter 5 for the linear case.

Lemma 3. *Assume $V_\theta(s_0)$ is continuously differentiable as a function of θ, $\forall s_0 \in \mathcal{S}$ s.t. $\mu(s_0) > 0$. Let $(S, \delta(\theta))$ be jointly distributed random variables, and assume that*

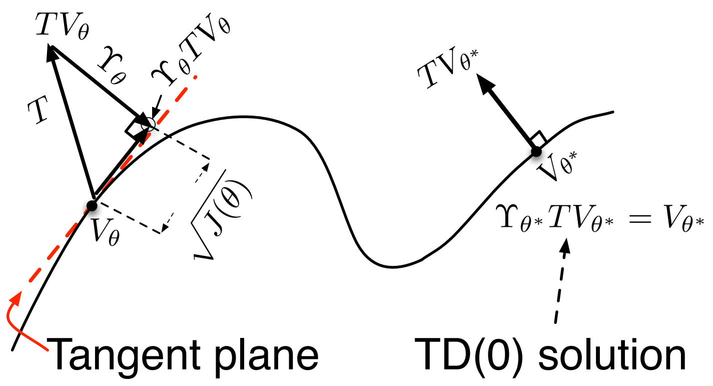

Figure 6.1: The MSPBE objective for nonlinear function approximation at two points in the value function space. The figure shows a point, V_θ, at which, $J(\theta)$, is not 0 and a point, V_{θ^*}, where $J(\theta^*) = 0$, thus $\Upsilon_{\theta^*} TV_{\theta^*} = V_{\theta^*}$, so this is a TD(0) solution.

$\mathbb{E}[\nabla V_\theta(S) \nabla V_\theta(S)^\top]$ *is nonsingular. Then*

$$J(\theta) \quad = \quad \mathbb{E}[\,\delta(\theta)\,\nabla V_\theta(S)\,]^\top\, \mathbb{E}[\,\nabla V_\theta(S)\nabla V_\theta(S)^\top\,]^{-1}\, \mathbb{E}[\,\delta(\theta)\,\nabla V_\theta(S)\,]. \tag{6.5}$$

Proof. The derivation of the identity is similar to that of MSPBE derivation for linear function approximation (see Equation 5.5 in Chapter 5), except that here Π_θ is expressed by (6.3), thus

$$\begin{aligned}
J(\theta) &= \|\, \Pi_\theta(TV_\theta - V_\theta)\,\|_\mu^2 \\
&= (\Pi_\theta(V_\theta - TV_\theta))^\top D(\Pi_\theta(V_\theta - TV_\theta)) \\
&= (V_\theta - TV_\theta)^\top \Pi_\theta^\top D \Pi_\theta (V_\theta - TV_\theta) \\
&= (V_\theta - TV_\theta)^\top D^\top \Phi_\theta(\Phi_\theta^\top D \Phi_\theta)^{-1} \Phi_\theta^\top D(V_\theta - TV_\theta) \\
&= (\Phi_\theta^\top D(TV_\theta - V_\theta))^\top (\Phi_\theta^\top D \Phi)^{-1} \Phi_\theta^\top D(TV_\theta - V_\theta) \\
&= \mathbb{E}[\,\delta(\theta)\,\nabla V_\theta(S)\,]^\top\, \mathbb{E}[\,\nabla V_\theta(S)\nabla V_\theta(S)^\top\,]^{-1}\, \mathbb{E}[\,\delta(\theta)\,\nabla V_\theta(S)\,],
\end{aligned}$$

where the Φ_θ matrix in defined by $(\Phi_\theta)_{s,i} = \frac{\partial}{\partial \theta_i} V_\theta(s)$.

□

Note that the assumption that $\mathbb{E}[\,\nabla V_\theta(S)\nabla V_\theta(S)^\top\,]^{-1}$ is non-singular is akin to the assumption that the feature vectors are independent in the linear function approximation case. We make this assumption here for convenience; it can be lifted, but the proofs become more involved.

Corollary 1. *Under the conditions of Lemma 3, $J(\theta) = 0$, if and only if V_θ is a TD(0) solution (i.e., if and only if it satisfies* (2.9)*).*

This is an important corollary, because it shows that optimizing the proposed objective function will indeed produce TD(0) solutions.

6.3 Derivation of nonlinear GTD2/TDC

To derive nonlinear GTD2/TDC algorithm, we now proceed to compute the gradient of MSPBE (6.5).

Theorem 4. *Assume that* (i) $V_\theta(s_0)$ *is twice continuously differentiable in* θ *for any* $s_0 \in \mathcal{S}$ *s.t.* $\mu(s_0) > 0$ *and* (ii) $W(\cdot)$ *defined by* $W(\hat{\theta}) = \mathbb{E}[\nabla V_{\hat{\theta}}(S)\,\nabla V_{\hat{\theta}}(S)^\top]$ *is non-singular in a small neighbourhood of* θ. *Let* $(S, \delta(\theta))$ *be jointly distributed random variables. Let* $\phi \equiv \nabla V_\theta(S)$, $\phi' \equiv \nabla V_\theta(S')$ *and*

$$h(\theta, u) \overset{\text{def}}{=} \mathbb{E}[\,(\delta(\theta) - \phi^\top u)\,\nabla^2 V_\theta(S) u\,], \tag{6.6}$$

where $u \in \mathbb{R}^d$. *Then*

$$\begin{aligned} &-\frac{1}{2}\nabla J(\theta) \\ &= \mathbb{E}[(\phi - \gamma\phi')\phi^\top w(\theta)] - h(\theta, w(\theta)) \qquad (6.7)\\ &= \mathbb{E}[\delta(\theta)\phi] - \gamma\mathbb{E}[\phi'\phi^\top w(\theta)] - h(\theta, w(\theta)), \qquad (6.8)\end{aligned}$$

where $w(\theta) = \mathbb{E}[\phi\,\phi^\top]^{-1}\,\mathbb{E}[\delta(\theta)\phi]$.

The main difference between Equation (6.8) and Equation (5.9), which shows the gradient for the linear case, is the appearance of the term $h(\theta, w)$, which involves second-order derivatives of V_θ (which are zero when V_θ is linear in θ).

Proof. The conditions of Lemma 3 are satisfied, so (6.5) holds. Denote $\partial_i = \frac{\partial}{\partial\theta_i}$. From its definition and the assumptions, $W(u)$ is a symmetric, positive definite matrix, so $\frac{d}{du}(W^{-1})|_{u=\theta} = -\,W^{-1}(\theta)\;(\frac{d}{du}W|_{u=\theta})\;W^{-1}(\theta)$, where we use the assumption that $\frac{d}{du}W$ exists at θ and W^{-1} exists in a small neighborhood of θ.

For the sake of simplicity, here, let us define $\delta \equiv \delta(\theta)$ and $w = w(\theta)$, thus, from the above identity, we have:

$$\begin{aligned}
&-\frac{1}{2}\left[\nabla J(\theta)\right]_i \\
&= -(\partial_i \mathbb{E}[\delta\phi])^\top \mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta\phi] - \frac{1}{2}\,\mathbb{E}[\delta\phi]^\top\, \partial_i \left(\mathbb{E}[\phi\phi^\top]^{-1}\right)\, \mathbb{E}[\delta\phi] \\
&= -(\partial_i \mathbb{E}[\delta\phi])^\top \mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta\phi] + \frac{1}{2}\,\mathbb{E}[\delta\phi]^\top\, \mathbb{E}[\phi\phi^\top]^{-1}(\partial_i \mathbb{E}[\phi\phi^\top])\, \mathbb{E}[\phi\phi^\top]^{-1}\, \mathbb{E}[\delta\phi] \\
&= -\mathbb{E}[\partial_i(\delta\phi)]^\top (\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta\phi]) \\
&\quad +\frac{1}{2}\,(\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta\phi])^\top\, \mathbb{E}[\partial_i(\phi\phi^\top)]\, (\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta\phi]).
\end{aligned}$$

The interchange between the gradient and expectation is possible here because of assumptions *(i)* and *(ii)* and the fact that $\mathcal{S}$ is finite. Now consider the identity

$$\frac{1}{2}x^\top \partial_i(\phi\phi^\top)x = \phi^\top x\ (\partial_i\phi^\top)x,$$

which holds for any vector $x \in \mathbb{R}^d$. Hence, using the definition of w,

$$\begin{aligned}
&-\frac{1}{2}\left[\nabla J(\theta)\right]_i \\
&= -\mathbb{E}[\partial_i(\delta\phi)]^\top w + \frac{1}{2}w^\top \mathbb{E}[\partial_i(\phi\phi^\top)]w \\
&= -\mathbb{E}[(\partial_i\delta)\phi^\top w] - \mathbb{E}[\delta(\partial_i\phi^\top)w] + \mathbb{E}[\phi^\top w(\partial_i\phi^\top)w].
\end{aligned}$$

Using $\nabla\delta = \gamma\phi' - \phi$ and $\nabla\phi^\top = \nabla^2 V_\theta(s)$, we get

$$\begin{aligned}
&-\frac{1}{2}\nabla J(\theta) \\
&= -\mathbb{E}[(\gamma\phi' - \phi)\phi^\top w] - \mathbb{E}[(\delta - \phi^\top w)\nabla^2 V(s)w] \\
&= \mathbb{E}[(\phi - \gamma\phi')\phi^\top w] - \mathbb{E}[(\delta - \phi^\top w)\nabla^2 V(s)w].
\end{aligned}$$

Finally, observe that

$$\begin{aligned}
&\mathbb{E}[(\phi - \gamma\phi')\phi^\top w] \\
&= \mathbb{E}[(\phi - \gamma\phi')\phi]^\top (\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta\phi]) \\
&= \mathbb{E}[\delta\phi] - \mathbb{E}[\gamma\phi'\phi^\top w],
\end{aligned}$$

concluding the proof. □

Using the expression derived in Theorem 4, it suggests the following generalization of linear GTD2/TDC (cf. Equations 5.8a and 5.10), to the nonlinear case. The weight w_k is updated as before on the "faster" timescale:

$$w_{k+1} = w_k + \beta_k(\delta_k - \phi_k^\top w_k)\phi_k. \tag{6.9}$$

The parameter vector θ_k is updated on a "slower" timescale, either according to

$$\theta_{k+1} = \Gamma\Big(\theta_k + \alpha_k \Big\{(\phi_k - \gamma\phi'_k)(\phi_k^\top w_k) - h_k\Big\}\Big), \qquad \text{(non-linear GTD2)} \tag{6.10}$$

or, according to

$$\theta_{k+1} = \Gamma\Big(\theta_k + \alpha_k \Big\{\delta_k\phi_k - \gamma\phi'_k(\phi_k^\top w_k) - h_k\Big\}\Big), \qquad \text{(non-linear TDC),} \tag{6.11}$$

where

$$h_k = (\delta_k - \phi_k^\top w_k)\, \nabla^2 V_{\theta_k}(s_k) w_k. \tag{6.12}$$

Besides h_k, the only new ingredient compared to the linear case is $\Gamma : \mathbb{R}^d \to \mathbb{R}^d$; a mapping that projects its argument into a set C, which is a parameter of the algorithm. Normally, one selects C to be a bounded set that has a smooth boundary and which is large enough so that the set of TD(0) solutions,

$$U = \{\, \theta \,|\, \mathbb{E}[\, \delta(\theta)\, \nabla V_\theta(S)] = 0 \,\},$$

is subsumed by C. The purpose of this projection step is to prevent the algorithms' parameters from divergence in the initial phase. Without the projection step this could happen due to the presence of the nonlinearities in the algorithm. Note that the projection is a common technique for stabilizing the transient behavior of stochastic approximation algorithms (Kushner and Yin, 2003). In practice, one selects C just large enough (by using *a priori* bounds on the size of the rewards and the derivative of the value function) in which case it is very likely that the parameter vector will not get projected at all during the execution of the algorithm. We also emphasize that the main reason for the projection is to facilitate convergence analysis. In many applications, this may not be needed at all.

Let us now analyze the computational complexity of these algorithms per update. Here we assume that we can compute $V_\theta(s)$ and its gradient with the cost of O(d) computation which is normally the case (e.g., neural networks). If the product of the Hessian of $V_\theta(s)$ and w can be computed in $O(d)$ time in (6.12), we immediately see that the computational cost of these algorithms per update will be $O(d)$. We show this is normally the case including neural networks. In the case of neural networks, let $V_\theta(s) = \sigma(\theta^\top x(s))$, where $\sigma(a) = \frac{1}{1+\exp(-a)}$, then $\nabla V_\theta(s) = [V_\theta(s)(1 - V_\theta(s))]\, x$ and

$$\nabla^2 V_\theta(s) w = \Big[V_\theta(s)\, (1 - V_\theta(s))\, (1 - 2V_\theta(s))\, x^\top w\Big]\, x.$$

The product of Hessian of $V_{\theta_k}(s)$ and w_k in (6.12) generally can be written as

$$\nabla^2 V_{\theta_k}(s) w_k = \nabla(\nabla V_{\theta_k}(s)^\top w_k),$$

because w_k does not depend on θ_k. As a result, because the scalar term, $\nabla V_{\theta_k}(s)^\top w_k$, costs only $O(d)$ to compute its gradient which is a vector and also has O(d) complexity. In general the observation that the product of a Hessian matrix and a vector of size d can be computed with the cost of O(d) is due to Pearlmutter (1994).

6.4 Convergence Analysis

In this section we provide a two-timescale convergence analysis (see Borkar, 2008) for nonlinear TDC– note, the analysis will be similar for nonlinear GTD2 as well.

Let $\mathcal{C}(\mathbb{R}^d)$ be the space of $\mathbb{R}^d \to \mathbb{R}^d$ continuous functions. Define the operator $\hat{\Gamma} : \mathcal{C}(\mathbb{R}^d) \to \mathcal{C}(\mathbb{R}^d)$ by

$$\hat{\Gamma} v\,(\theta) = \lim_{0<\varepsilon\to 0} \frac{\Gamma\big(\theta + \varepsilon\, v(\theta)\big) - \theta}{\varepsilon}.$$

In fact, because by assumption $\Gamma(\theta) = \arg\min_{\theta' \in C} \|\theta' - \theta\|$ and the boundary of C is smooth, $\hat{\Gamma}$ is well defined and in particular $\hat{\Gamma} v\,(\theta) = v(\theta)$ when $\theta \in C^\circ$ and otherwise $\hat{\Gamma} v\,(\theta)$ is the projection of $v(\theta)$ to the tangent space of ∂C at $\Gamma(\theta)$.

Consider the ODE

$$\dot{\theta} = \hat{\Gamma}(-\tfrac{1}{2}\nabla J)(\theta). \tag{6.13}$$

Let K be the set of all asymptotically stable fixed points of (6.13). By its definition, $K \subset C$. Further, for $U \cap C \subset K$ (i.e., if θ is a TD(0)-solution that lies in C then it is an asymptotically stable fixed point of (6.13)).

The next theorem shows that under some technical conditions, the iterates produced by nonlinear TDC converge to K with probability one. Thus, apart from the projection step, the algorithm converges to the stationary points of the objective function J, which is the best result one can in general hope when using a stochastic gradient algorithm with a non-convex objective function.

Theorem 5 (Convergence of nonlinear TDC)**.** *Let $(S_k, R_k, S'_k)_{k\geq 0}$ be a sequence of i.i.d. transitions that satisfies (A1). Consider the nonlinear TDC iterations* (6.9), (6.11). *With positive deterministic step-size sequences that satisfy $\sum_{k=0}^\infty \alpha_k = \sum_{k=0}^\infty \beta_k = \infty$, $\sum_{k=0}^\infty \alpha_k^2$, $\sum_{k=0}^\infty \beta_k^2 < \infty$ and $\frac{\alpha_k}{\beta_k} \to 0$, as $k \to \infty$. Assume that for each $s_0 \in \mathcal{S}$ such that $\mu(s_0) > 0$, for all $\theta \in C$, $V_\theta(s_0)$ is three times continuously differentiable. Further assume that for* *each $\theta \in C$, $\mathbb{E}[\nabla V_\theta(S) \nabla V_\theta(S)^\top]$ is nonsingular. Then $\theta_k \to K$, with probability one, as $k \to \infty$.*

Proof. Let (S_k, R_k, S'_k) be a random transition whose law is the same as the law underlying $(S_k, R_k, S'_k)_{k \geq 0}$. Further, let $\phi_\theta = \nabla V_\theta(S)$, $\phi'_\theta = \nabla V_\theta(S')$, $\phi_k = \nabla V_{\theta_k}(S_k)$, and $\phi'_k = \nabla V_{\theta_k}(S'_k)$.

We begin by rewriting the updates (6.9)-(6.11) as follows:

$$w_{k+1} = w_k + \beta_k(f(\theta_k, w_k) + M_{k+1}), \tag{6.14}$$

$$\theta_{k+1} = \Gamma\big(\theta_k + \alpha_k(g(\theta_k, w_k) + N_{k+1})\big), \tag{6.15}$$

where

$$\begin{aligned} f(\theta_k, w_k) &= \mathbb{E}[\delta_k \phi_k | \theta_k] - \mathbb{E}[\phi_k \phi_k^\top | \theta_k] w_k, \\ g(\theta_k, w_k) &= \mathbb{E}\Big[\delta_k \phi_k - \gamma \phi'_k \phi_k^\top w_k - h_k | \theta_k, w_k\Big], \end{aligned}$$

$$\begin{aligned} M_{k+1} &= (\delta_k - \phi_k^\top w_k)\phi_k - f(\theta_k, w_k), \\ N_{k+1} &= (\delta_k \phi_k - \gamma \phi'_k \phi_k^\top w_k - h_k) - g(\theta_k, w_k). \end{aligned}$$

We need to verify that there exists a compact set $B \subset \mathbb{R}^{2d}$ such that (a) the functions $f(\theta, w)$, $g(\theta, w)$ are Lipschitz continuous over B (b) $(M_k, \mathcal{G}_k)$, $(N_k, \mathcal{G}_k)$, $k \geq 0$ are martingale difference sequences, where $\mathcal{G}_k = \sigma(R_i, \theta_i, w_i, S_i, i \leq k; S'_i, i < k)$, $k \geq 0$ are increasing σ-fields, (c) $\{(w_k(\theta), \theta)\}$ with $\phi_k(\theta) = \nabla V_\theta(S_k)$ and $w_k(\theta)$ update

$$w_{k+1}(\theta) = w_k(\theta) + \beta_k\Big(\delta_k(\theta) - \phi_k(\theta)^\top w_k(\theta)\Big)\phi_k(\theta),$$

almost surely stays in B for any choice of $(w_0(\theta), \theta) \in B$, and (d) $\{(w, \theta_k)\}$ almost surely stays in B for any choice of $(w, \theta_0) \in B$. From these, thanks to the conditions on the step-sizes, standard arguments (c.f. Theorem 2 of Sutton et al., 2009; Borkar, 1997; Borkar 2008; Szepesvári, 2010) allow us to deduce that θ_k almost surely converges to the set of asymptotically stable fixed points of

$$\dot{\theta} = \hat{\Gamma} F(\theta),$$

where $F(\theta) = g(\theta, w(\theta))$. Here for $\theta \in C$ fixed, $w(\theta)$ is the (unique) equilibrium point of

$$\dot{w}(\theta) = \mathbb{E}[\delta(\theta)\phi_\theta] - \mathbb{E}[\phi_\theta \phi_\theta^\top] w(\theta), \tag{6.16}$$

where $\delta(\theta) = R + \gamma V_\theta(S') - V_\theta(S)$. Clearly, $w(\theta) = \mathbb{E}\big[\phi_\theta \phi_\theta^\top\big]^{-1} \mathbb{E}[\delta(\theta)\phi_\theta]$, which exists by assumption. Then by Theorem 4 it follows that $F(\theta) = -\frac{1}{2}\nabla J(\theta)$. Hence, the statement will follow once (a)–(d) are verified.

Note that (a) is satisfied because V_θ is three times continuously differentiable. For (b), we need to verify that $\mathbb{E}[M_{k+1} \mid \mathcal{G}_k] = 0$, $\mathbb{E}[N_{k+1} \mid \mathcal{G}_k] = 0 \; \forall k \geq 0$, which in fact follow from the definitions. Condition (c) follows since, by a standard argument (e.g., Borkar & Meyn (2000)), $w_k(\theta)$ converges to $w(\theta)$, which by assumption stays bounded if θ comes from a bounded set. Now for condition (d), note that $\{\theta_k\}$ is uniformly bounded since $\theta_k \in C$, $\forall k$, and by assumption C is a compact set. □

6.5 Empirical results

To illustrate the convergence properties of the algorithms, we applied them to the "spiral" counterexample mentioned in Section 2.7, originally used to show the divergence of TD(0) with nonlinear function approximation. The Markov chain with 3 states is shown in the Figure 6.2. The reward is always zero and the discount factor is $\gamma = 0.9$. The value function has a single parameter, θ, and takes the nonlinear spiral form

$$V_\theta(s) = \Big(a(s)\cos(\hat{\lambda}\theta) - b(s)\sin(\hat{\lambda}\theta)\Big) e^{\epsilon\theta}.$$

The true value function is $V = (0,0,0)^\top$ which is achieved as $\theta \to -\infty$. Here we used $V_0 = (100, -70, -30)^\top$, $a = V_0$, $b = (23.094, -98.15, 75.056)^\top$, $\hat{\lambda} = 0.866$ and $\epsilon = 0.05$. Note that this is a degenerate example, in which our theorems do not apply, because the optimal parameter values are infinite. Hence, we run our algorithms without a projection step.

We also use constant learning rates, in order to facilitate gradient descent through an error surface which is essentially flat. For TDC we used $\alpha = 0.5$, $\beta = 0.05$, and for GTD2, $\alpha = 0.8$ and $\beta = 0.1$. For TD(0) we used $\alpha = 2 \times 10^{-3}$ (as argued by Tsitsiklis & Van Roy (1997), tuning the step-size does not help with the divergence problem). All step sizes are then normalized by $\|V_\theta^\top D \frac{d}{d\theta} V_\theta\|$.

Figure 6.2, shows the performance measure, $\sqrt{J}$, as a function of the number of updates (we used expected updates for all the algorithms). GTD2 and TDC converge to the correct

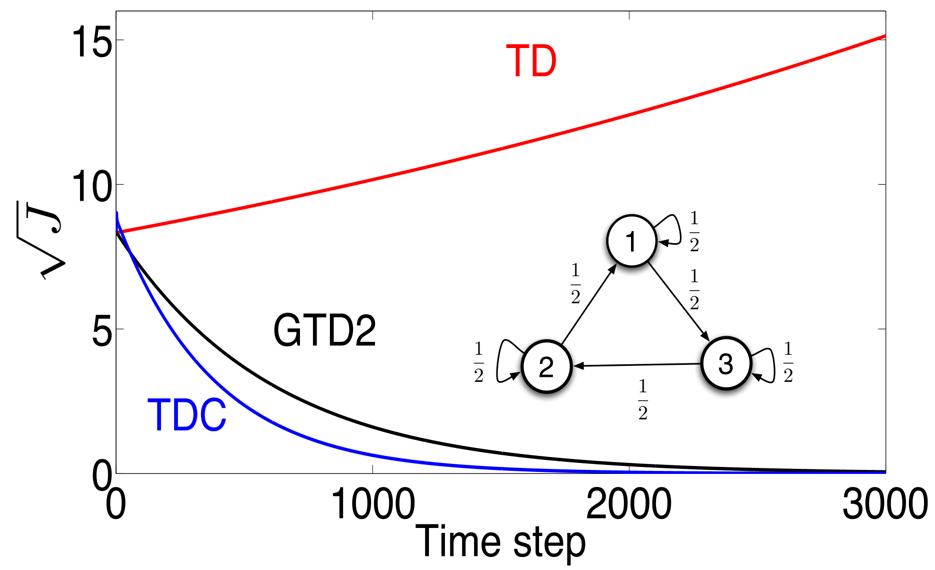

Figure 6.2: Empirical evaluation results for spiral counterexample.

solution, while TD(0) diverges. We note that convergence happens despite the fact that this example is outside the scope of the theory.

To assess the performance of the new algorithms on a large scale problem, David Silver used them to learn an evaluation function in 9x9 computer Go. He used a version of RLGO (Silver, 2009), in which a logistic function is fit to evaluate the probability of winning from a given position. Positions were described using 969,894 binary features corresponding to all possible shapes in every 3x3, 2x2, and 1x1 region of the board. Using weight sharing to take advantage of symmetries, the million features were reduced to a parameter vector of $d = 63,303$ components. Experience was generated by self-play, with actions chosen uniformly randomly among the legal moves. All rewards were zero, except upon winning the game, when the reward was 1. David Silver applied the four algorithms to this problem: TD(0), the proposed algorithms (GTD2 and TDC) and residual gradient (RG). In the experiments, RG was run with only one sample.

In each run, θ was initialized to random values uniformly distributed in $[-0.1, 0.1]$; for GTD2 and TDC, the second parameter vector, w, was initialized to 0. Training then proceeded for 5000 complete games, after which θ was frozen. This problem is too large to compute the objective function J. Instead, to assess the quality of the solutions obtained, he estimated the average prediction error of each algorithm. More precisely, he generated 2500 test games; for every state occurring in a game, he computed the squared error between its

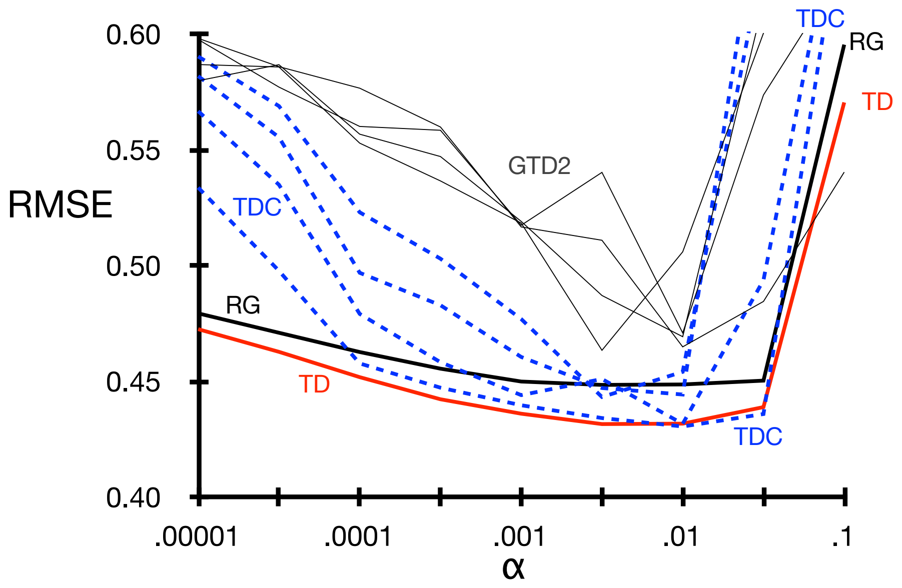

Figure 6.3: Empirical evaluation on 9x9 Computer Go with nonlinear function approximation.

predicted value and the actual return that was obtained in that game. He then computed the root of the mean-squared error, averaged over all time steps. Figure 6.3 plots this measure over a range of values of the learning rate α. The results are averages over 50 independent runs. For TDC and GTD he used several values of the β parameter, which generate the different curves. As was noted in previous empirical work, TD provides slightly better estimates than the RG algorithm. TDC's performance is very similar to TD, for a wide range of parameter values. GTD2 is slightly worse. These results are very similar in flavour to those obtained in Section 5.4 using the same domain, but with linear function approximation.

6.6 Conclusions

In this chapter, we solved a long-standing open problem in reinforcement learning, by establishing a family of temporal-difference learning algorithms that converge with arbitrary differentiable function approximators (including neural networks). The algorithms perform gradient descent on a natural objective function, the projected Bellman error. The local optima of this function coincide with solutions that could be obtained by TD(0). Of course, TD(0) may not converge with non-linear function approximation. Our algorithms are on-line, incremental and their computational cost per update is linear in the number of parameters. Our theoretical results guarantee convergence to a local optimum, under standard technical assumptions. Local optimality is the best one can hope for, since nonlinear function approximation creates non-convex optimization problems. The early empirical results obtained for computer Go shows nonlinear TDC is more effective than GTD/GTD2, which is similar to our linear results in the previous Chapter.

Chapter 7

Extension to Eligibility Traces

One of the key properties of TD methods is their ability to learn from successive predictions. However, this may come with a cost of being sensitive to changes in successive predictions. Monte-Carlo methods, on the other hand, do not have such sensitivity as the error is only between the current prediction and the final outcome. To bridge the gap between one-step TD prediction and the Monte-Carlo method, eligibility traces are incorporated in TD methods. Eligibility traces are essential for TD prediction because they bridge the temporal gaps in cause and effect when experience is processed at a temporally fine resolution. Eligibility traces, allow us to look further ahead through their mechanistic backward-view updates. Several important properties of eligibility traces are as follows:

- They make TD methods more like efficient incremental Monte-Carlo algorithms. For example, in TD(λ), $\lambda \in [0, 1]$ refers to eligibility function, and is equivalent to Monte-Carlo methods when $\lambda = 1$.
- They are particularly of interest when reward is delayed by many steps, thus, by adjusting λ function we may get faster and efficient learning.

To learn more about how the notion of eligibility traces appears in TD learning we refer the reader to the background section and also Sutton & Barto (1998).

In the previous chapters, we presented gradient TD algorithms that were based on one-step TD prediction; that is, their predictions were only based on one-step look ahead. In this chapter, we extend our gradient TD methods to include eligibility traces. Particularly, we extend the TDC algorithm, which is used for state-value function predictions, to include eligibility traces, and we call the resulting algorithm GTD(λ).

7.1 Problem formulation and objectives

Without loss of generality we use linear value function approximation— similar analysis can be used for the nonlinear setting. We define the λ-return (function) according to

$$G_t^\lambda(V) \overset{\text{def}}{=} R_{t+1} + \gamma \left[(1-\lambda)V(S_{t+1}) + \lambda G_{t+1}^\lambda\right],$$

where $V(S_{t+1})$ denotes value function at state S_{t+1}, and $\lambda \in [0,1]$ is a constant eligibility trace parameter. For the table-look up case, a λ-weighted version of the Bellman equation follows from MDP property, which can be written as:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}\Big[G_t^\lambda(V^\pi) \mid S_t = s, \pi\Big]. \\ &\overset{\text{def}}{=} (T^{\pi\lambda}V^\pi)(s), \end{aligned}$$

where $T^{\pi\lambda}$ is called the λ-weighted Bellman operator for policy π. Note, the notation $\mathbb{E}[.|s,\pi]$ indicates that the expectation is over the data that is generated by following policy π from state s. For detailed description of λ-weighted Bellman equation we refer the reader to Tsitsiklis and Van Roy (1997).

Let us consider linear function approximation: $V(s) \approx V_\theta(s) = \theta^\top \phi(s)$. Our objective is to find off-policy TD-solution, θ, which satisfies $V_\theta = \Pi T^{\pi\lambda} V_\theta$ (Π is defined in Equation (2.15)), while the data is generated according to a behavior policy π_b, with state distribution μ, that is, $s \sim \mu(.)$.

Objective function Just like in Section 5.2, consider the following mean-square projected Bellman-error (MSPBE) objective function:

$$J(\theta) = \|V_\theta - \Pi T^{\pi\lambda} V_\theta\|_\mu^2. \tag{7.1}$$

Let us, first, consider the following definitions and identities. We start with the following definitions:

$$\phi_t \equiv \phi(S_t),$$

$$G_t^\lambda(\theta) \overset{\text{def}}{=} R_{t+1} + \gamma \left[(1-\lambda)\theta^\top \phi_{t+1} + \lambda G_{t+1}^\lambda(\theta)\right], \tag{7.2}$$

$$\delta_t^\lambda(\theta) \overset{\text{def}}{=} G_t^\lambda(\theta) - \theta^\top \phi_t, \tag{7.3}$$

and

$$\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t \quad \overset{\text{def}}{=} \quad \sum_s \mu(s)\mathbb{E}\Big[\delta_t^\lambda(\theta)|S_t = s, \pi\Big]\, \phi(s), \tag{7.4}$$

where $\mathcal{P}_\mu^\pi$ is an operator.

And now consider the following identities:

$$\mathbb{E}\Big[\delta_t^\lambda(\theta)|S_t = s, \pi\Big] = (T^{\pi\lambda}V_\theta - V_\theta)(s),$$

$$\begin{aligned}
\mathbb{E}\Big[\phi_t\phi_t^\top\Big] &= \sum_s \mu(s)\phi(s)\phi(s)^\top \\
&= \Phi^\top D \Phi,
\end{aligned}$$

$$\begin{aligned}
\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t &= \sum_s \mu(s)\mathbb{E}\Big[\delta_t^\lambda(\theta)|S_t = s, \pi\Big]\, \phi(s) \\
&= \sum_s \mu(s)\, \Big[(T^{\pi\lambda}V_\theta - V_\theta)(s)\Big]\, \phi(s) \\
&= \Phi^\top D \Big(T^{\pi\lambda}V_\theta - V_\theta\Big),
\end{aligned}$$

where D and Φ are defined in Sec. 2.5.

Following derivations in Chapter 5, which led to Equation (5.5), and also definitions and identities, which we introduced above, we have:

$$\begin{aligned}
J(\theta) &= \|V_\theta - \Pi T^{\pi\lambda}V_\theta\|_\mu^2 \\
&= (\Phi^\top D(T^{\pi\lambda}V_\theta - V_\theta))^\top(\Phi^\top D\Phi)^{-1}\Phi^\top D(T^{\pi\lambda}V_\theta - V_\theta) \\
&= (\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t)^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1}(\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t). \qquad (7.5)
\end{aligned}$$

Practically, there are two major issues with the objective function (7.5): 1) The expectation term is with respect to the policy π, while the data is generated according to behavior policy

π_b, 2) We can't do forward-view equations as they need future data.

To overcome the first issue, we use importance weighting scenario (also see Sec 4.2). In the next section, we show how to use importance weighting, and show a forward-view objective function whose expectation terms are with respect to behavior policy π_b. Later, we use the mechanistic TD backward-view to deal with the second issue.

7.2 Forward-view objective function

The expectation TD update term in $J(\theta)$, is with respect to target policy π. In order to convert it to π_b, we need to use the notion of importance sampling. After we conduct this step, we will show how to transform the forward-view TD update terms into mechanistic backward-view.

Importance weighting scenario Consider an agent which takes action, A, according to behavior policy π_b. We would like to estimate the return (future rewards) if the agent were to follow target policy π. We use the following recursive $\lambda-$return equation, based on importance sampling, at time t:

$$G_t^{\lambda\rho}(\theta) = \rho_t \left(r_{t+1} + \gamma \left[(1-\lambda)\theta^\top \phi_{t+1} + \lambda G_{t+1}^{\lambda\rho}(\theta) \right] \right),$$

where

$$\rho_t = \frac{\pi(A_t \mid S_t)}{\pi_b(A_t \mid S_t)}.$$

$G_t^{\lambda\rho}(\theta)$ is a noisy guess of future rewards from policy π, while following policy π_b. Let

$$\delta_t^{\lambda\rho}(\theta) \quad = \quad G_t^{\lambda\rho}(\theta) - \theta^\top \phi_t. \tag{7.6}$$

Theorem 6. *(Off-policy TD with importance weighting) Let π_b and π denote the behavior and target policies, respectively. Consider $\delta_t^\lambda(\theta)$, $\delta_t^{\lambda\rho}(\theta)$ defined in equations* (7.3) (7.6)*. Then,*

$$\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t = \mathbb{E}\left[\delta_t^{\lambda\rho}(\theta)\phi_t \right]. \tag{7.7}$$

Proof. We show this by expanding the right-hand side

$$\begin{aligned}
&\mathbb{E}\Big[G_t^{\lambda\rho}(\theta) \mid S_t = s\Big] \\
&= \mathbb{E}\Big[\rho_t \Big(R_{t+1} + \gamma(1-\lambda)\theta^\top \phi_{t+1}\Big) + \rho_t \gamma\lambda G_{t+1}^{\lambda\rho}(\theta) \mid S_t = s\Big] \\
&= \mathbb{E}\Big[\rho_t \Big(R_{t+1} + \gamma(1-\lambda)\theta^\top \phi_{t+1}\Big) \mid S_t = s\Big] + \rho_t\gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\rho}(\theta) \mid S_t = s\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top \phi_{t+1} \mid S_t = s, \pi\Big] \\
&\quad + \sum_{a,s'} P(s' \mid s,a)\pi_b(a \mid s)\frac{\pi(a|s)}{\pi_b(a \mid s)}\gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\gamma\rho}(\theta) \mid S_{t+1} = s'\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top \phi_{t+1} \mid S_t = s, \pi\Big] \\
&\quad + \sum_{a,s'} P(s' \mid s,a)\pi(a \mid s)\gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\gamma\rho}(\theta) \mid S_{t+1} = s'\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top \phi_{t+1} + \gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\gamma\rho}(\theta) \mid S_{t+1} = s'\Big] \mid S_t = s, \pi\Big],
\end{aligned}$$

which, as it continues to roll out, gives us

$$\mathbb{E}\Big[G_t^{\lambda\rho}(\theta) \mid S_t = s\Big] = \mathbb{E}\Big[G_t^{\lambda}(\theta) \mid S_t = s, \pi\Big].$$

And, eventually we get:

$$\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] = \mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t,$$

because the state-distribution is based on behavior state-distribution, μ. □

The forward-view objective function (7.5), now can be written based on the importance weighting idea, that is:

$$\begin{aligned}
&J(\theta) \\
&= \Big(\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t\Big)^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1} \Big(\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t\Big) \\
&= \mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]. \qquad (7.8)
\end{aligned}$$

7.3 Backward-view objective function

In this section, we transform the TD forward-view to mechanistic backward-view. To do this, we propose the following theorem, which provides a great tool for forward-view to backward-view transformation.

Theorem 7. *(Equivalence of the TD forward-view and backward-view) The forward-view description of TD update is equivalence to the following mechanistic backward-view:*

$$\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] = \mathbb{E}[\delta_t(\theta)e_t]\,, \tag{7.9}$$

where $\delta_t^{\lambda\rho}(\theta)$ is defined in Equation (7.6), $\delta_t(\theta)$ is the conventional TD error, $\delta_t(\theta) = R_{t+1} + \gamma\theta^\top\phi_{t+1} - \theta^\top\phi_t$, and e_t is the eligibility trace vector at time-step t, and has the following recursive update:

$$e_t = \rho_t\left(\phi_t + \gamma\lambda e_{t-1}\right). \tag{7.10}$$

Proof. Consider

$$\begin{aligned}
G_t^{\lambda\rho}(\theta) &= \rho_t\Big(R_{t+1} + \gamma\Big[(1-\lambda)\theta^\top\phi_{t+1} + \lambda G_{t+1}^{\lambda\rho}(\theta)\Big]\Big) \\
&= \rho_t\Big(R_{t+1} + \gamma\theta^\top\phi_{t+1} - \theta^\top\phi_t + \theta^\top\phi_t\Big) - \rho_t\gamma\lambda\theta^\top\phi_{t+1} + \rho_t\gamma\lambda G_{t+1}^{\lambda\gamma\rho}(\theta) \\
&= \rho_t\Big(R_{t+1} + \gamma\theta^\top\phi_{t+1} - \theta^\top\phi_t\Big) + \rho_t\theta^\top\phi_t + \rho_t\gamma\lambda\Big(G_{t+1}^{\lambda\gamma\rho}(\theta) - \theta^\top\phi_{t+1}\Big) \\
&= \rho_t\delta_t(\theta) + \rho_t\theta^\top\phi_t + \rho_t\gamma\lambda\delta_{t+1}^{\lambda\gamma\rho}(\theta),
\end{aligned}$$

thus,

$$\begin{aligned}
\delta_t^{\lambda\rho}(\theta) &= G_t^{\lambda\rho}(\theta) - \theta^\top\phi_t \\
&= \rho_t\delta_t(\theta) + \rho_t\theta^\top\phi_t + \rho_t\gamma\lambda\delta_{t+1}^{\lambda\rho}(\theta) - \theta^\top\phi_t \\
&= \rho_t\delta_t(\theta) + (\rho_t - 1)\theta^\top\phi_t + \rho_t\gamma\lambda\delta_{t+1}^{\lambda\rho}(\theta).
\end{aligned}$$

Also, consider the following identity:

$$\begin{aligned}
\mathbb{E}\Big[(1-\rho_t)\theta^\top\phi_t\phi_t\Big] &= \sum_{s,a}\mu(s)\pi_b(a\mid s)\left(1 - \frac{\pi(a\mid s)}{\pi_b(a\mid s)}\right)\theta^\top\phi(s)\phi(s) \\
&= \sum_s\mu(s)\left(\sum_a\pi_b(a\mid s) - \sum_a\pi(a\mid s)\right)\theta^\top\phi(s)\phi(s) \\
&= \sum_s\mu(s)\,(1-1)\,\theta^\top\phi(s)\phi(s) \\
&= 0,
\end{aligned}$$

and consequently, $\mathbb{E}\big[(1-\rho_t)\theta^\top\phi_t\phi_k\big] = 0$, for $k < t$.

Putting all above together, we get:

$$\begin{aligned}
&\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] \\
&= \mathbb{E}\Big[\rho_t\delta_t(\theta)\phi_t + (\rho_t-1)\theta^\top\phi_t + \rho_t\gamma\lambda\delta_{t+1}^{\lambda\rho}(\theta)\phi_t\Big] \\
&= \mathbb{E}[\rho_t\delta_t(\theta)\phi_t] + 0 + \mathbb{E}_{\pi_b}\Big[\rho_t\gamma\lambda\delta_{t+1}^{\lambda\rho}(\theta)\phi_t\Big] \\
&= \mathbb{E}\Big[\rho_t\delta_t(\theta)\phi_t + \rho_{t-1}\gamma\lambda\delta_t^{\lambda\rho}\phi_{t-1}\Big] \\
&= \mathbb{E}\Big[\delta_t^{\gamma\rho}(\theta)\phi_t + \rho_{t-1}\gamma\lambda\Big(\rho_t\delta_t(\theta)\phi_t + (\rho_t-1)\theta^\top\phi_t + \rho_t\gamma\lambda\delta_{t+1}^{\lambda\rho}(\theta)\phi_t\Big)\phi_{t-1}\Big] \\
&= \mathbb{E}\Big[\rho_t\delta_t(\theta)\phi_t + \rho_{t-1}\gamma\lambda\Big(\rho_t\delta_t(\theta) + \rho_t\gamma\lambda\delta_{t+1}^{\lambda\rho}(\theta)\Big)\phi_{t-1}\Big] \\
&= \mathbb{E}\Big[\rho_t\delta_t(\theta)\left(\phi_t + \rho_{t-1}\gamma\lambda\phi_{t-1}\right) + \rho_{t-1}\gamma\lambda\rho_t\gamma\lambda\delta_{t+1}^{\lambda\rho}(\theta)\phi_{t-1}\Big] \\
&= \mathbb{E}\Big[\rho_t\delta_t(\theta)\left(\phi_t + \gamma\lambda\rho_{t-1}\phi_{t-1}\right) + \rho_{t-2}\gamma\lambda\rho_t\gamma\lambda\delta_t^{\lambda\rho}(\theta)\phi_{t-2}\Big] \\
&\vdots \\
&= \mathbb{E}[\delta_t(\theta)\rho_t\left(\phi_t + \rho_{t-1}\gamma\lambda\phi_{t-1} + \rho_{t-2}\gamma\lambda\rho_{t-1}\gamma\lambda\phi_{t-2} + \cdots\right)] \\
&= \mathbb{E}[\delta_t(\theta)e_t]\,, \qquad (7.11)
\end{aligned}$$

where $e_t = \rho_t(\phi_t + \gamma\lambda e_{t-1})$. $\square$

7.4 Derivation of the GTD(λ) algorithm

Now from Equation (7.8) and Theorem 7, we get

$$J(\theta) = \mathbb{E}[\delta_t(\theta)e_t]^\top\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\,.$$

Just like the TDC derivation, we have:

$$\begin{aligned}
&-\frac{1}{2}\nabla J(\theta) \\
&= -\frac{1}{2}\nabla\Big(\mathbb{E}[\delta_t(\theta)e_t]^\top\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\Big) \\
&= -\nabla\mathbb{E}\Big[\delta_t(\theta)e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= -\mathbb{E}\Big[\left(\gamma\phi_{t+1}-\phi_t\right)e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\,.
\end{aligned}$$

We use the following identity:

$$\begin{aligned}
\mathbb{E}\Big[\phi_t e_t^\top\Big] &= \mathbb{E}\Big[\phi_t \rho_t (\phi_t + \gamma\lambda e_{t-1})^\top\Big] \\
&= \mathbb{E}\Big[\phi_t \rho_t \phi_t^\top + \phi_t \gamma\lambda e_{t-1}^\top\Big] \\
&= \mathbb{E}\Big[\phi_t \rho_t \phi_t^\top + \phi_{t+1} \rho_t \gamma\lambda e_t^\top\Big] \\
&= \mathbb{E}\Big[\phi_t \phi_t^\top + \phi_{t+1} \gamma\lambda e_t^\top\Big],
\end{aligned}$$

where we have used shifting indices trick and the following identities

$$\mathbb{E}\Big[\phi_t \rho_t \phi_t^\top\Big] = \mathbb{E}\Big[\phi_t \phi_t^\top\Big], \quad \mathbb{E}\Big[\phi_{t+1} \rho_t \gamma\lambda e_t^\top\Big] = \mathbb{E}\Big[\phi_{t+1} \gamma\lambda e_t^\top\Big].$$

Thus,

$$\begin{aligned}
-\mathbb{E}\Big[(\gamma\phi_{t+1} - \phi_t)\, e_t^\top\Big] &= \mathbb{E}\Big[\gamma\phi_{t+1} e_t^\top - \phi_t e_t^\top\Big] \\
&= -\mathbb{E}\Big[\gamma\phi_{t+1} e_t^\top - \Big(\phi_t \phi_t^\top + \phi_{t+1}\gamma\lambda e_t^\top\Big)\Big] \\
&= \mathbb{E}\Big[\phi_t \phi_t^\top - \gamma(1-\lambda)\phi_{t+1} e_t^\top\Big].
\end{aligned}$$

Using the above identity, we get

$$\begin{aligned}
&-\frac{1}{2}\nabla J(\theta) \\
&= -\mathbb{E}\Big[(\gamma\phi_{t+1} - \phi_t)\, e_t^\top\Big]\, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta) e_t] \\
&= \mathbb{E}\Big[\phi_t\phi_t^\top - \gamma(1-\lambda)\phi_{t+1} e_t^\top\Big]\, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta) e_t] \\
&= \mathbb{E}[\delta_t(\theta) e_t] - \mathbb{E}\Big[\gamma(1-\lambda)\phi_{t+1} e_t^\top\Big]\, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta) e_t] \\
&= \mathbb{E}[\delta_t(\theta) e_t] - \mathbb{E}\Big[\gamma(1-\lambda)\phi_{t+1} e_t^\top\Big]\, w(\theta), \qquad (7.12)
\end{aligned}$$

where $w(\theta) = \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta) e_t]$.

The GTD(λ) algorithm By direct sampling from Equation (7.12) and following the TDC derivations steps we get the GTD(λ) algorithm:

$$\theta_{t+1} = \alpha_t \left[\delta_t e_t - \gamma(1-\lambda)(e_t^\top w_t)\phi_{t+1}\right], \qquad (7.13a)$$

$$w_{t+1} = \beta_t \left[\delta_t e_t - (w_t^\top \phi_t)\phi_t\right], \qquad (7.13b)$$

where

$$\delta_t = r_{t+1} + \gamma_{t+1}\theta_t^\top \phi_{t+1} - \theta_t^\top \phi_t,$$

$$e_t = \rho_t \left(\phi_t + \gamma\lambda e_{t-1}\right),$$

$$\rho_t = \frac{\pi(A_t \mid S_t)}{\pi_b(A_t \mid S_t)}.$$

Algorithm 1 GTD(λ) with linear function approximation

```
1: **Initialize** $w_0$ to 0, and $\theta_0$ arbitrarily.
2: **Choose** proper (small) positive values for $\alpha$, $\beta$, and set values for $\gamma \in (0,1]$, $\lambda \in [0,1]$.

3: **Repeat** for each episode:
4: **Initialize** $e = 0$.
5: **Take** $A_t$ from $S_t$ according to $\pi_b$, and arrive at $S_{t+1}$.
6: **Observe** sample, $(\phi_t, R_{t+1}, \phi_{t+1})$ at time step $t$.
7: **for** each observed sample **do**
8:     $\delta_t \leftarrow R_{t+1} + \gamma\theta_t^\top \phi_{t+1} - \theta_t^\top \phi_t$.
9:     $\rho_t \leftarrow \frac{\pi(A_t|S_t)}{\pi_b(A_t|S_t)}$.
10:    $e_t \leftarrow \rho_t \left(\phi_t + \gamma\lambda e_{t-1}\right)$.
11:    $\theta_{t+1} \leftarrow \theta_t + \alpha_t \left[\delta_t e_t - \gamma(1-\lambda)(e_t^\top w_t)\phi_{t+1}\right]$.
12:    $w_{t+1} \leftarrow w_t + \beta_t \left[\delta_t e_t - (\phi_t^\top w_t)\phi_t\right]$.
13: **end for**
```

Convergence remarks: Just like the convergence remarks in Sub-section 5.3.4 for TDC with importance weighting, it can be shown that GTD(λ) is guaranteed to converge under standard conditions and proper choice of step-sizes.

7.5 Conclusion

In this chapter we extended our gradient-TD algorithm (TDC-style update) to include eligibility traces. To derive the GTD(λ) algorithm we carried out a forward-view/backward-view analysis and used importance sampling. Just like our previous gradient-TD algorithms, a convergence guarantee can be assured for GTD(λ). A through convergence analysis, with Markov noise, can be carried out as a future work by following the remarks in Sub-section 5.3.4.

Chapter 8

GQ(λ): A Gradient-TD Algorithm for General Prediction

In this chapter[1], we develop a new off-policy TD method, called GQ(λ), for estimating action-value functions. This new algorithm can be seen as extension of GTD(λ), which was developed for estimating state-value functions, to an off-policy TD method for estimating action-value functions. Because action-value functions typically are denoted by Q, thus we call our new algorithm GQ(λ). GQ(λ) can be thought of as an extension of Q-learning as well. We extend our existing convergence results for policy evaluation to this setting. Just like GTD(λ), we derive GQ(λ) by conducting forward-view/backward-view analysis. Our analysis for deriving GQ(λ) is similar to that of GTD(λ) with some subtle differences in definition of λ-return, which results in different updates. Later in this chapter, we show how to extend the GQ(λ) algorithm to the control domain.

8.1 Problem formulation and objectives

We consider the problem of policy evaluation in finite state-action Markov Decision Process (MDP) and standard RL setting. Let

$$Q^{\pi}(s,a) \overset{\text{def}}{=} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a, \pi\right],$$

where $\gamma \in (0,1]$ is discount factor and $Q^{\pi}(s,a)$ denotes action-value function that evaluates policy π given state-action pair (s,a). To simplify the notation, from now on, we drop the superscript π on action values.

[1]This chapter is based on the following paper: Maei, H. R. and Sutton, R. S. (2010). GQ(λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the Third Conference on Artificial General Intelligence*, pp. 91–96. Atlantis Press.

Analogous to state-value functions, we define the λ-return for action-value functions:

$$G_t^\lambda(Q) \;=\; R_{t+1} + \gamma \Big[(1-\lambda)Q(S_{t+1}, A_{t+1}) + \lambda G_{t+1}^\lambda(Q)\Big], \tag{8.1}$$

where $Q(s,a)$ denotes the value of taking action a from state s, $\gamma \in (0,1]$, and $\lambda \in [0,1]$. Under MDP assumption, for every entry of state-action pair we get the following λ-weighted Bellman equation for action-value functions:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi\Big[G_t^\lambda(Q^\pi) \mid S_t = s, A_t = a, \pi\Big]. \\ &\overset{\text{def}}{=} (T^{\pi\lambda}Q^\pi)(s,a), \end{aligned}$$

where $T^{\pi\lambda}$ is a λ-weighted state–action version of the affine $|\mathcal{S}\times\mathcal{A}| \times |\mathcal{S}\times\mathcal{A}|$ Bellman operator for the target policy π.

To estimate action-value functions, we use linear function approximation, where $Q \approx Q_\theta = \Phi\theta$ is the vector of approximate action values for each state–action pair $(s,a) \in \mathcal{S}\times\mathcal{A}$, and Φ is the matrix whose rows are the state–action feature vectors $\phi(s,a)^\top \in \mathbb{R}^d$

Our objective is to estimate Q^π action-value functions under the following constraint: (1) Linear function approximation, (2) Data is generated according to behavior policy π_b, (3) TD learning with linear complexity both in terms of memory and per-time-step complexity. Here, the TD solution satisfies $Q_\theta = \Pi T^{\pi\lambda} Q_\theta$, where $\Pi = \Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D$, is a projection matrix that projects any point in the action-value space into the linear space of approximate action values, and D is a diagonal matrix whose diagonal (s,a) entries correspond to the frequency, $\mu(s,a)$, with which each state–action pair is visited under the behavior policy.

Following our approach for derivation of GTD(λ) in Section 7.1, we consider the following projected Bellman-error objective function:

$$\begin{aligned} J(\theta) &= \|Q_\theta - \Pi T^{\pi\lambda} Q_\theta\|_\mu^2 \\ &= \Big(\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t\Big)^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1} \Big(\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t\Big), \end{aligned} \tag{8.2}$$

where δ_t^λ denotes *λ-weighted TD-error* at time t:

$$\delta_t^\lambda(\theta) \overset{\text{def}}{=} G_t^\lambda(\theta) - \theta^\top\phi_t, \tag{8.3}$$

$\mathcal{P}_\mu^\pi$ operator is defined in Equation (7.4), $G_t^\lambda(\theta)$ is

$$G_t^\lambda(\theta) \quad = \quad R_{t+1} + \gamma \left[(1-\lambda)\theta^\top \phi_{t+1} + \lambda G_{t+1}^\lambda(\theta) \right], \tag{8.4}$$

and $\phi_t \equiv \phi(S_t, A_t)$.

First, in the next section we introduce the GQ(λ) algorithm, a gradient TD method, whose learning parameter is updated according stochastic gradient descent in the objective function $J(\theta)$. Later, in this chapter, we show the derivation of algorithm..

8.2 The GQ(λ) algorithm

First, we specify the GQ(λ) algorithm as follows: The weight vector $\theta \in \mathbb{R}^d$ is initialized arbitrarily. The secondary weight vector $w \in \mathbb{R}^d$ is initialized to zero. An auxiliary memory vector known as the eligibility trace $e \in \mathbb{R}^d$ is also initialized to zero. Their update rules are

$$\theta_{t+1} \quad = \quad \theta_t + \alpha_t \Big[\delta_t e_t - \gamma(1-\lambda)(w_t^\top e_t)\bar{\phi}_{t+1} \Big], \tag{8.5a}$$

$$w_{t+1} \quad = \quad w_t + \beta_t \left[\delta_t e_t - (w_t^\top \phi_t)\phi_t \right], \tag{8.5b}$$

and

$$e_t = \phi_t + \gamma\lambda\rho_t e_{t-1}, \tag{8.6}$$

where,

$$\delta_t = R_{t+1} + \gamma\theta_t^\top \bar{\phi}_{t+1} - \theta_t^\top \phi_t, \tag{8.7}$$

$$\bar{\phi}_t = \sum_a \pi(a \mid S_t)\phi(S_t, a), \tag{8.8}$$

$$\rho_t = \frac{\pi(A_t \mid S_t)}{\pi_b(A_t \mid S_t)},$$

where ϕ_t is an alternate notation for $\phi(S_t, A_t)$, and $\alpha_t > 0$, $\beta_t > 0$, are positive step-size parameters for θ and w weights respectively.

In the next section we derive GQ(λ) based on gradient-descent in projected (λ-weighted) Bellman error objective function.

Algorithm 2 GQ(λ) with linear function approximation

```
1: Initialize $w_0$ to 0, and $\theta_0$ arbitrarily.
2: Choose proper (small) positive values for $\alpha$, $\beta$, and set values for $\gamma \in (0,1]$, $\lambda \in [0,1]$.

3: Repeat for each episode:
4: Initialize $e = 0$.
5: Take $A_t$ from $S_t$ according to $\pi_b$, and arrive at $S_{t+1}$.
6: Observe sample, $(S_t, R_{t+1}, S_{t+1})$ at time step $t$ (with their corresponding state-action
   feature vectors).
7: for each observed sample do
8:     $\bar{\phi}_{t+1} \leftarrow \sum_a \pi(a \mid S_{t+1})\phi(S_{t+1}, a)$.
9:     $\delta_t \leftarrow R_{t+1} + \gamma\theta_t^\top \bar{\phi}_{t+1} - \theta_t^\top \phi_t$.
10:    $\rho_t \leftarrow \frac{\pi(A_t|S_t)}{\pi_b(A_t|S_t)}$.
11:    $e_t \leftarrow \phi_t + \rho_t\gamma\lambda e_{t-1}$.
12:    $\theta_{t+1} \leftarrow \theta_t + \alpha\left[\delta_t e_t - \gamma(1-\lambda)(e_t^\top w_t)\bar{\phi}_{t+1}\right]$.
13:    $w_{t+1} \leftarrow w_t + \beta\left[\delta_t e_t - (\phi_t^\top w_t)\phi_t\right]$.
14: end for
```

8.3 Derivation of GQ(λ)

To derive GQ(λ), we follow the GTD(λ) derivation steps; that is: (1) Importance weighting and deriving forward-view objective function whose expectation terms are with respect to behavior policy π_b, (2) Transforming the TD forward-view update to backward-view, and as a result obtaining the backward-view objective function, (3) Updating learning parameters along stochastic gradient-descent of desired objective function.

8.3.1 Forward-view objective function based on importance sampling

The expectation terms, in the proposed objective function (8.2), are with respect to the target policy π while the data is generated according to behavior policy π_b. To overcome this issue, we use importance sampling. We start with equation (8.1), which is a noisy estimate of future rewards (return) by following policy π. To have a noisy estimate for the return of policy π while following policy π_b, we define the following $\lambda-$weighted return function:

$$G_t^{\lambda\rho}(\theta) = R_{t+1} + \gamma\left[(1-\lambda)\theta^\top \bar{\phi}_{t+1} + \lambda\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta)\right],$$

where

$$\bar{\phi}_t = \sum_a \pi(a \mid S_t)\phi(S_t, a),$$

$$\rho_t = \frac{\pi(A_t \mid S_t)}{\pi_b(A_t \mid S_t)},$$

and $G_t^{\lambda\rho}(\theta)$ is a noisy guess of future rewards of policy π, if the agent follows policy π from time t.

Let

$$\delta_t^{\lambda\rho}(\theta) \quad = \quad G_t^{\lambda\rho}(\theta) - \theta^\top \phi_t. \tag{8.9}$$

The next Theorem shows $\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t$ (see Eq. 8.2) can be replaced by $\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]$.

Theorem 8. *(Off-policy TD with important weighting) Let π_b and π denote the behavior and target policies, respectively. Consider $\delta_t^\lambda(\theta)$, $\delta_t^{\lambda\rho}(\theta)$ defined in equations* (8.3) (8.9)*. Then,*

$$\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t = \mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]. \tag{8.10}$$

Proof. We show this by expanding the right-hand side

$$\begin{aligned}
&\mathbb{E}_{\pi_b}\Big[G_t^{\lambda\rho}(\theta) \mid S_t = s, A_t = a\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma\Big((1-\lambda)\theta^\top\bar{\phi}_{t+1} + \lambda\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta)\Big) \mid S_t = s, A_t = a\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top\bar{\phi}_{t+1} \mid S_t = s, A_t = a, \pi\Big] \\
&\qquad + \gamma\lambda\mathbb{E}\Big[\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta) \mid S_t = s, A_t = a\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top\bar{\phi}_{t+1} | S_t = s, A_t = a, \pi\Big] \\
&\qquad + \sum_{s'} P(s' \mid s, a) \sum_{a'} \pi_b(a' \mid s') \frac{\pi(a'|s')}{\pi_b(a' \mid s')} \gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\rho}(\theta) \mid S_{t+1} = s', A_{t+1} = a'\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top\bar{\phi}_{t+1} \mid S_t = s, A_t = a, \pi\Big] \\
&\qquad + \sum_{s',a'} P(s' \mid s, a)\pi(a' \mid s')\gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\rho} \mid S_{t+1} = s', A_{t+1} = a'\Big] \\
&= \mathbb{E}\Big[R_{t+1} + \gamma(1-\lambda)\theta^\top\bar{\phi}_{t+1} + \gamma\lambda\mathbb{E}\Big[G_{t+1}^{\lambda\rho}(\theta) \mid S_{t+1} = s', A_{t+1} = a'\Big] \mid S_t = s, A_t = a, \pi\Big],
\end{aligned}$$

which, as it continues to roll out, gives us

$$\mathbb{E}\Big[G_t^{\lambda\rho}(\theta) \mid S_t = s, A_t = a\Big] = \mathbb{E}\Big[G_t^\lambda(\theta) \mid S_t = s, A_t = a, \pi\Big].$$

And, eventually it yields:

$$\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] = \mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t,$$

because the state-action distribution is based on behavior state-action pair distribution, μ.

$\square$

Therefore, from Theorem 8, the forward-view objective function (8.2) can be written as:

$$\begin{aligned} & J(\theta) \\ = & \Big(\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t\Big)^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1} \Big(\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t\Big) \\ = & \mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] \end{aligned} \tag{8.11}$$

8.3.2 Backward-view objective function

In this section, we transform the TD forward-view to a mechanistic backward-view. To do this, we propose the following theorem, which provides a great tool for forward-view to backward-view transformation.

Theorem 9. *(Equivalence of TD forward-view and backward-view) The forward-view description of TD update is equivalence to the following mechanistic backward-view:*

$$\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] = \mathbb{E}[\delta_t(\theta)e_t]\,, \tag{8.12}$$

where $\delta_t^{\lambda\rho}(\theta)$ is defined in Equation (8.9), $\delta_t(\theta) = R_{t+1} + \gamma\theta^\top\bar{\phi}_{t+1} - \theta^\top\phi_t$, and e_t is eligibility trace vector at time-step t, and has the following recursive update:

$$e_t = \phi_t + \gamma\lambda\rho_t e_{t-1}. \tag{8.13}$$

Proof. Consider

$$\begin{aligned}
G_t^{\lambda\rho}(\theta) &= R_{t+1} + \gamma \left[(1-\lambda)\theta^\top \bar{\phi}_{t+1} + \lambda \rho_{t+1} G_{t+1}^{\lambda\rho}(\theta)\right] \\
&= \left[R_{t+1} + \gamma(1-\lambda)\theta^\top \bar{\phi}_{t+1}\right] + \gamma\lambda\rho_{t+1} G_{t+1}^{\lambda\rho}(\theta) \\
&= \left(R_{t+1} + \gamma\theta^\top \bar{\phi}_{t+1}\right) - \gamma\lambda\theta^\top \bar{\phi}_{t+1} + \gamma\lambda\rho_{t+1} G_{t+1}^{\lambda\rho}(\theta) \\
&= \left(R_{t+1} + \gamma\theta^\top \bar{\phi}_{t+1} - \theta^\top \phi_t + \theta^\top \phi_t\right) - \gamma\lambda\theta^\top \bar{\phi}_{t+1} + \gamma\lambda\rho_{t+1} G_{t+1}^{\lambda\rho}(\theta) \\
&= \left(\delta_t(\theta) + \theta^\top \phi_t\right) - \gamma\lambda\theta^\top \bar{\phi}_{t+1} + \gamma\lambda\rho_{t+1} G_{t+1}^{\lambda\rho}(\theta) + \gamma\lambda\rho_{t+1}\left(\theta^\top \phi_{t+1} - \theta^\top \phi_{t+1}\right) \\
&= \left(\delta_t(\theta) + \theta^\top \phi_t\right) + \gamma\lambda\rho_{t+1}\left(G_{t+1}^{\lambda\rho}(\theta) - \theta^\top \phi_{t+1}\right) + \gamma\lambda\left(\rho_{t+1}\theta^\top \phi_{t+1} - \theta^\top \bar{\phi}_{t+1}\right) \\
&= \left(\delta_t(\theta) + \theta^\top \phi_t\right) + \gamma\lambda\rho_{t+1}\delta_t^{\lambda\rho}(\theta) + \gamma\lambda\left(\rho_{t+1}\theta^\top \phi_{t+1} - \theta^\top \bar{\phi}_{t+1}\right),
\end{aligned}$$

thus,

$$\begin{aligned}
\delta_t^{\lambda\rho}(\theta) &= G_t^{\lambda\rho}(\theta) - \theta^\top \phi_t \\
&= \delta_t(\theta) + \gamma\lambda\rho_{t+1}\delta_t^{\lambda\rho}(\theta) + \gamma\lambda\left(\rho_{t+1}\theta^\top \phi_{t+1} - \theta^\top \bar{\phi}_{t+1}\right)
\end{aligned}$$

Note that the last part of the above equation has expected value of vector zero under the behavior policy because

$$\begin{aligned}
\mathbb{E}[\rho_t \phi_t \mid S_t] &= \sum_a \pi_b(a \mid S_t) \frac{\pi(a \mid S_t)}{\pi_b(a \mid S_t)} \phi(a \mid S_t) \\
&= \sum_a \pi(a \mid S_t)\phi(S_t, a) \equiv \bar{\phi}_t.
\end{aligned}$$

Putting all these together, we can write the TD update (in expectation) in a simple way in terms of eligibility traces which leads to backward-view:

$$
\begin{aligned}
&\mathbb{E}\Big[\delta_t^{\lambda\rho}\phi_t\Big]\\
&= \mathbb{E}\Big[\Big(\delta_t + \gamma\lambda\rho_{t+1}\delta_{t+1}^{\lambda\rho}\Big)\,\phi_t\Big] + \mathbb{E}\Big[\gamma\lambda\theta^\top\left(\rho_{t+1}\phi_{t+1} - \bar{\phi}_{t+1}\right)\phi_t\Big]\\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}\Big[\gamma\lambda\rho_{t+1}\delta_{t+1}^{\lambda\rho}\phi_t\Big] + 0\\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}\Big[\gamma\lambda\rho_t\delta_t^{\lambda\rho}\phi_{t-1}\Big]\\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}_b[\gamma\lambda\rho_t\Big(\delta_t + \gamma\lambda\rho_{t+1}\delta_{t+1}^{\lambda\rho} + \gamma\lambda\theta^\top\left(\rho_{t+1}\phi_{t+1} - \bar{\phi}_{t+1}\right)\Big)\phi_{t-1}]\\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}[\gamma\lambda\rho_t\delta_t\phi_{t-1}] + \mathbb{E}\Big[\gamma\lambda\rho_t\gamma\lambda\rho_{t+1}\delta_{t+1}^{\lambda\rho}\phi_{t-1}\Big] + 0\\
&= \mathbb{E}[\delta_t\left(\phi_t + \gamma\lambda\rho_t\phi_{t-1}\right)] + \mathbb{E}\Big[\gamma\lambda_{t-1}\rho_{t-1}\gamma\lambda\rho_t\delta_t^{\lambda\rho}\phi_{t-2}\Big]\\
&\vdots\\
&= \mathbb{E}_b\Big[\delta_t\Big(\phi_t + \gamma\lambda\rho_t\phi_{t-1} + \gamma\lambda\rho_t\gamma\lambda_{t-1}\rho_{t-1}\phi_{t-2} + \cdots\Big)\Big]\\
&= \mathbb{E}[\delta_t e_t]\,, \qquad (8.14)
\end{aligned}
$$

where $e_t = \phi_t + \gamma\lambda\rho_t e_{t-1}$, which gives us a backward view algorithm for the TD(λ) update.

$\square$

8.3.3 Stochastic gradient-descent derivation

Now from Equation (8.11) and Theorem 9, we get

$$J(\theta) = \mathbb{E}[\delta_t(\theta)e_t]^\top\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\,.$$

Following the derivation of GTD(λ), we get:

$$
\begin{aligned}
&-\frac{1}{2}\nabla J(\theta)\\
&= -\frac{1}{2}\nabla\Big(\mathbb{E}[\delta_t(\theta)e_t]^\top\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\Big)\\
&= -\nabla\mathbb{E}\Big[\delta_t(\theta)e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\\
&= -\mathbb{E}\Big[\left(\gamma\bar{\phi}_{t+1} - \phi_t\right)e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\,. \qquad (8.15)
\end{aligned}
$$

We use the following identity:

$$\begin{aligned}
\mathbb{E}\Big[\phi_t e_t^\top\Big] &= \mathbb{E}\Big[\phi_t(\phi_t + \gamma\lambda\rho_t e_{t-1})^\top\Big] \\
&= \mathbb{E}\Big[\phi_t\phi_t^\top + \gamma\lambda\rho_t\phi_t e_{t-1}^\top\Big] \\
&= \mathbb{E}\Big[\phi_t\phi_t^\top + \gamma\lambda\rho_{t+1}\phi_{t+1} e_t^\top\Big] \\
&= \mathbb{E}\Big[\phi_t\phi_t^\top + \gamma\lambda\bar{\phi}_{t+1} e_t^\top\Big],
\end{aligned}$$

where we have used $\mathbb{E}\big[\rho_{t+1}\phi_{t+1}e_t^\top\big] = \mathbb{E}\big[\bar{\phi}_{t+1}e_t^\top\big]$.

Thus, from above and Equation (8.15), we get

$$\begin{aligned}
&-\frac{1}{2}\nabla J(\theta) \\
&= -\mathbb{E}\Big[\big(\gamma\bar{\phi}_{t+1} - \phi_t\big)\, e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= -\mathbb{E}\Big[\gamma\bar{\phi}_{t+1}e_t^\top - \phi_t e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= -\mathbb{E}\Big[\gamma\bar{\phi}_{t+1}e_t^\top - \Big(\phi_t\phi_t^\top + \gamma\lambda\bar{\phi}_{t+1}e_t^\top\Big)\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= -\mathbb{E}\Big[\gamma(1-\lambda)\bar{\phi}_{t+1}e_t^\top - \phi_t\phi_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= \mathbb{E}[\delta_t(\theta)e_t] - \mathbb{E}\Big[\gamma(1-\lambda)\bar{\phi}_{t+1}e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= \mathbb{E}[\delta_t(\theta)e_t] - \mathbb{E}\Big[\gamma(1-\lambda)\bar{\phi}_{t+1}e_t^\top\Big]\, w(\theta), \qquad (8.16)
\end{aligned}$$

where $w(\theta) = \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]$.

Thus, by direct sampling from the above gradient-descent direction and weight-duplication trick we get:

$$\theta_{t+1} = \theta_t + \alpha_t\Big[\delta_t e_t - \gamma(1-\lambda)\bar{\phi}_{t+1}(e_t^\top w_t)\Big],$$

$$w_{t+1} = w_t + \beta_t\left[\delta_t e_t - (w_t^\top\phi_t)\phi_t\right],$$

Convergence remarks: Convergence analysis of GQ(λ) is similar to GTD(λ). Thus, see the convergence remarks for GTD(λ) in Section 7.4.

8.4 Greedy-GQ(λ): The extension of GQ(λ) to control

Just like the extension of TD(0) (with action values) to Q-learning (Watkins, 1989), we can also extend GQ(λ), which was developed for the problem of prediction, to control. Specifically, we consider the target policy π_θ to be greedy with respect to Q_θ. As a result, we get the Greedy-GQ(λ) algorithm. Thus, the form of the Greedy-GQ(λ) iterates are the same as GQ(λ)'s iterates, and the only difference is that now the fixed target policy π is substituted with (parametrized) varying target policy π_θ .

Algorithm (3), shows how to use the Greedy-GQ(λ) algorithm. Note, for the case of $\lambda = 0$,

Algorithm 3 Greedy-GQ(λ) with linear function approximation

```
1: **Initialize** $w_0$ to 0, and $\theta_0$ arbitrarily.
2: **Choose** proper (small) positive values for $\alpha$, $\beta$, and set values for $\gamma \in (0,1]$, $\lambda \in [0,1]$.

3: **Repeat** for each episode:
4: **Initialize** $e = 0$.
5: **Take** $A_t$ from $S_t$ according to $\pi_b$, and arrive at $S_{t+1}$.
6: **Observe** sample, $(S_t, R_{t+1}, S_{t+1})$ at time step $t$ (with their corresponding state-action
   feature vectors), where $\hat{\phi}_{t+1}(\theta_t) = \phi(S_{t+1}, A^*_{t+1})$, $A^*_{t+1} = \text{argmax}_b\, \theta_t^\top \phi(S_{t+1}, b)$.
7: **for** each observed sample **do**
8:    $\delta_t \leftarrow R_{t+1} + \gamma \theta_t^\top \hat{\phi}_{t+1}(\theta) - \theta_t^\top \phi_t$.
9:    If $A_t = A^*_t$, then $\rho_t \leftarrow \frac{1}{\pi_b(A^*_t|S_t)}$; otherwise $\rho_t \leftarrow 0$.
10:   $e_t \leftarrow \phi_t + \rho_t \gamma \lambda e_{t-1}$.
11:   $\theta_{t+1} \leftarrow \theta_t + \alpha_t \left[\delta_t e_t - \gamma(1-\lambda)(\phi_t^\top w_t)\hat{\phi}_{t+1}(\theta_t)\right]$.
12:   $w_{t+1} \leftarrow w_t + \beta_t \left[\delta_t e_t - (\phi_t^\top w_t)\phi_t\right]$.
13: **end for**
```

we call GQ(0), the Greedy-GQ algorithm. Greedy-GQ has the following update rules:

$$\theta_{t+1} = \theta_t + \alpha_t \left[\delta_t \phi_t - \gamma(\phi_t^\top w_t)\phi(S_{t+1}, A^*_{t+1})\right], \tag{8.17a}$$

$$w_{t+1} = w_t + \beta_t \left[\delta_t \phi_t - (\phi_t^\top w_t)\phi_t\right], \tag{8.17b}$$

where

$$\delta_t = R_{t+1} + \gamma \max_b \left(\theta_t^\top \phi(S_{t+1}, b)\right) - \theta_t^\top \phi(S_t, A_t),$$

and

$$A^*_{t+1} = \underset{b}{\text{argmax}} \left(\theta_t^\top \phi(S_{t+1}, b)\right).$$

Clearly, the first term of θ-update, is the same as Q-learning update with linear function approximation.

Maei et al. (2010) show that under a fixed behavior policy, Greedy-GQ is convergent while Q-learning can diverge (Baird, 1995; 1999). An stronger proof, with milder assumptions, can be done by using Theorem 17 in page 239 of Benveniste et al. (1990) (and also results presented in Delyon, 1996).

8.4.1 Empirical results

In this sub-section, we illustrate the (empirical) convergence result of Greedy-GQ on a well known off-policy example; Baird's counterexample (Baird, 1995), for which Q-learning diverges. This has been demonstrated in Figure 8.1.

Here, we have used the 7-star version of the "star" counterexample (see also Section 2.7). The MDP consists of 7 states and 2 actions for each state. The reward is always zero and the discount factor is $\gamma = 0.99$. In this problem, the true action value is zero for all state–action pairs. The initial value of θ parameters for the action that causes transition to the 7^{th} state is $(1, 1, 1, 1, 1, 1, 10, 1)$ and the rest are 1. The initial values for auxiliary weights w were set to zero.

Updating was done synchronously in dynamic-programming-like sweeps through the state-action space. The step-size parameter $\alpha = 0.1$ was used for Q-learning, and for Greedy-GQ we used $\alpha = 0.05$, $\alpha_w = 0.25$. Figure 8.1 shows how the performance measure, $\sqrt{J}$ (root of the MSPBE objective function), evolves with respect to the number of updates. Both algorithms used expected updates. The graph shows that Greedy-GQ finds the optimal

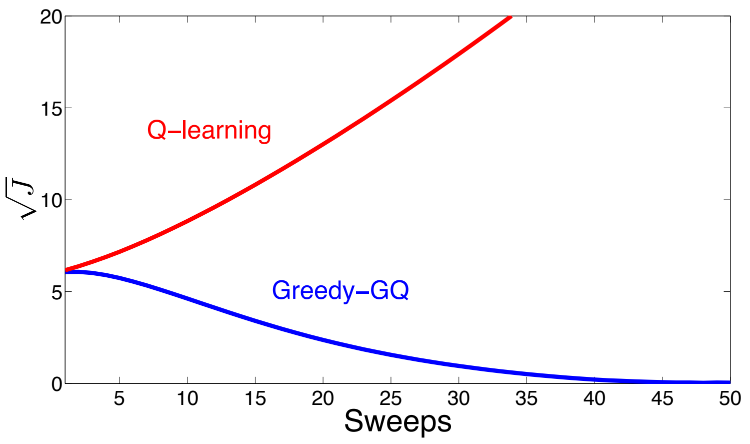

Figure 8.1: Empirical illustration for Baird's counterexample. The graph shows that Greedy-GQ converges to the true solution, while Q-learning diverges.

weights, while Q-learning diverges.

8.5 Conclusion

The GQ(λ) algorithm, which has been introduced in this chapter, incorporates eligibility traces into gradient-TD algorithms (TDC-style update) for estimating action-value functions. To derive GQ(λ), we carried out a forward-view/backward-view analysis. GQ(λ) is guaranteed to converge using our existing convergence results. GQ(λ) is a general gradient TD method for off-policy learning and as such can be seen as extension of Q-learning. GQ(λ) is online, incremental and its computational complexity scales only linearly with the size of features. Our work, however, is limited to policy evaluation. In this thesis mainly we have considered the problem of prediction. However, GQ(λ) can be extended to control domains, as we mentioned briefly.

Chapter 9

General Value Functions

In this chapter[1], we extend GQ(λ) and GTD(λ), which we developed in the previous chapters, to a more general settings, including general value functions (GVFs), terminal-reward functions (outcomes), and allow policies to terminate at any given state with a termination (probability) function. The GVFs are introduced in Sutton et al. (2011) (also see Maei & Sutton, 2010). Under the MDP property, GVFs satisfy a Bellman equation and thus we can use gradient TD methods to learn their values.

9.1 GVFs: The framework

We consider the problem of policy evaluation in finite state-action Markov Decision Process (MDP). Under standard conditions, however, our results can be extended to MDPs with infinite state–action pairs. In the standard RL setting, data is obtained from a continually evolving MDP with states $S_t \in \mathcal{S}$, actions $A_t \in \mathcal{A}$, and rewards $R_t \in \mathbb{R}$, for $t = 1, 2, \ldots$, with each state and reward as a function of the preceding state and action. Actions are chosen according to the behavior policy π_b, which is assumed fixed and exciting, $\pi_b(a \mid s) > 0, \forall(s, a)$; for the case of policy evaluation. We consider the transition probabilities between state–action pairs, and for simplicity we assume there is a finite number of state–action pairs.

Suppose that the agent finds itself in a state–action pair S_t, A_t, at time t. The agent expects its action-value function at that time, to tell something about the future sequence, $S_{t+1}, A_{t+1}, \ldots, S_{t+k}$, if actions were taken according to the policy, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, until its termination time $t + k$. The termination probability of policy π for a given state s is $1 - \gamma(s)$, where $\gamma : \mathcal{S} \to [0, 1]$.

[1]This chapter is based on the following paper: Maei, H. R. and Sutton, R. S. (2010). GQ(λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the Third Conference on Artificial General Intelligence*, pp. 91–96. Atlantis Press.

In standard RL, for episodic tasks, γ function normally is constant and is equal to 1, while for continuing task (infinite horizon), normally it is considered to be constant and less than 1.

The next step, is to answer what we are trying to predict. In standard RL, the most common type of prediction is the expected total or discounted future reward, while following a policy. However, rewards could also be represented in the form of transient signals while acting—transients are a measure of what happens during the trajectory rather than its end. We denote the transient signal $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ (note, we will show random variable with capital letters).

The second type of prediction is the expected outcome of policy upon its termination. We call this function the *outcome target function*, or *terminal-reward function*, $z : \mathcal{S} \rightarrow \mathbb{R}$, where $z(s)$ is the outcome that the agent receives if the policy terminates at state s.

Finally, the prediction could conceivably be a mixture of both a transient *and* an outcome. Here we will present the algorithm for predictions with both an outcome part z and a transient part r, with the two added together. In the common place where one wants only one of the two, the other is set to zero.

Now we can start to state the goal of learning more precisely. In particular, we would like our prediction to be equal to the expected value of the outcome target function at termination plus the cumulative sum of the transient reward function along the way. Thus, conventional action-value functions are defined in the following general form

$$Q^{\pi}(s,a) \equiv \mathbb{E}[R_{t+1} + R_{t+2} + \cdots + R_{t+k} + Z_{t+k} \mid S_t = s, A_t = a, \pi, \gamma] , \qquad (9.1)$$

where $\gamma : \mathcal{S} \rightarrow [0,1]$ is a discount function representing the probability of continuing to evaluate policy π from a given state, $Q^{\pi}(s,a)$ denotes action value function that evaluates policy π given state-action pair s, a, and its termination probability $1 - \gamma(s)$. We call these action-value functions, general value functions (GVFs). To simplify the notation, from now on, we drop the superscript π on action-values.

Our definition of GVFs does not involve discounting factor, as policies will terminate after some finite time—due to termination probability function $1 - \gamma$. Later we will see that γ function can be interpreted as discounting factor—$\gamma(s)$ indicates probability of continuing policy, π, from state s.

The Equation (9.1) describes the value functions in a Monte Carlo sense, but of course we want to include the possibility of temporal-difference learning. To do this, first we re-write the Equation (9.1) in the following bootstrapping form, which is derived under MDP assumption:

$$\begin{aligned}
&Q^\pi(s,a) \\
&= \mathbb{E}[R_{t+1} + R_{t+2} + \cdots + R_{t+k} + z_{t+k} \mid S_t = s, A_t = a, \pi, \gamma] \\
&= \sum_{s'} P(s' \mid s,a) \left[r(s,a,s') + (1-\gamma(s'))z(s') + \gamma(s') \sum_{a'} \pi(a' \mid s') Q^\pi(s',a') \right] \\
&\overset{\text{def}}{=} (T^\pi Q)(s,a), \qquad (9.2)
\end{aligned}$$

where T^π has the form of Bellman operator for any given state-action pair, thus, we call the above equation a Bellman equation.

Now that we have the above bootstrapping equation, we can implement our gradient TD methods, which we developed in previous sections, to this setting. Here, we use linear function approximation; that is, $Q_\theta(s,a) = \theta^\top \phi(s,a) \approx Q^\pi(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, where $\theta \in \mathbb{R}^d$ is a learned weight vector and $\phi(s,a) \in \mathbb{R}^d$ indicates a state–action feature vector. We want to learn parameter vector θ through a gradient TD method.

In the next section, we show how GQ(λ), which we developed in Chapter (8), can be extended to a more general settings, including GVFs, and varying eligibility traces; that is, instead of a constant eligibility trace parameter, we let eligibility trace function, for a given state, to be an arbitrary function of that state. It is easy to see that all of our convergence analysis will be extended to such general settings, and similar studies can be done for estimating (general) state-value functions using GTD(λ) developed in Chapter (7).

9.2 GQ(λ) for GVFs

In this section we show how to use GQ(λ) for learning parameters of GVFs. To do this, we consider four functions: π and γ, for the policy, and z and r, for the target functions. To specify how the answers will be formed, one provides their functional form (here in linear form). Here, the feature vectors are denoted $\phi(s,a)$ for all state–action pairs, and the eligibility-trace function is λ. The discount factor, which typically is used in standard RL setting can be taken to be 1, and thus ignored— the same effect as discounting can be achieved through the choice of γ function.

Now, we specify the GQ(λ) algorithm for GVFs as follows: The weight vector $\theta \in \mathbb{R}^d$ is initialized arbitrarily. The secondary weight vector $w \in \mathbb{R}^d$ is initialized to zero. An auxiliary memory vector known as the eligibility trace $e \in \mathbb{R}^d$ is also initialized to zero.

GQ(λ) update rules are

$$
\begin{aligned}
\theta_{t+1} &= \theta_t + \alpha_t\Big[\delta_t e_t - \gamma_t(1-\lambda_t)(w_t^\top e_t)\bar{\phi}_{t+1}\Big], && (9.3a)\\
w_{t+1} &= w_t + \beta_t\Big[\delta_t e_t - (w_t^\top \phi_t)\phi_t\Big], && (9.3b)\\
e_t &= \phi_t + \gamma_t \lambda_t \rho_t e_{t-1}, && (9.3c)\\
\delta_t &= R_{t+1} + (1-\gamma_{t+1})Z_{t+1} + \gamma_{t+1}\theta_t^\top \bar{\phi}_{t+1} - \theta_t^\top \phi_t, && (9.3d)\\
\bar{\phi}_t &= \sum_a \pi(a \mid S_t)\phi(a \mid S_t), && (9.3e)\\
\rho_t &= \frac{\pi(A_t \mid S_t)}{\pi_b(A_t \mid S_t,)}, && (9.3f)
\end{aligned}
$$

ϕ_t and γ_t are an alternate notations for $\phi(S_t, A_t)$ and $\gamma(S_t)$, respectively. Here, δ_t is one-step TD error, and the sequence $(\alpha_t, \beta_t; t \geq 0)$ are positive constant or decreasing step-size parameters.

Table (4) shows how to use GQ(λ) with linear function approximation for GVFs.

Algorithm 4 GQ(λ) for GVFs

```
1:  Initialize $w_0$ to 0, and $\theta_0$ arbitrarily.
2:  Choose proper (small) positive values for $\alpha$, $\beta$.
3:  Repeat for each episode:
4:  Initialize $e = 0$.
5:  Take $A_t$ from $S_t$ according to $\pi_b$, and arrive at $S_{t+1}$.
6:  Observe sample, $(S_t, R_{t+1}, Z_{t+1}, S_{t+1})$ at time step $t$ (with their corresponding state-
    action feature vectors).
7:  for each observed sample do
8:      $\bar{\phi}_{t+1} \leftarrow \sum_a \pi(a \mid S_{t+1})\phi(S_{t+1}, a)$, and
        $\lambda_t \leftarrow \lambda(S_t)$, $\gamma_t \leftarrow \gamma(S_t)$,$\gamma_{t+1} \leftarrow \gamma(S_{t+1})$.
9:      $\delta_t \leftarrow \big(R_{t+1} + (1-\gamma_{t+1})Z_{t+1} + \gamma_{t+1}\theta_t^\top \bar{\phi}_{t+1}\big) - \theta_t^\top \phi_t$.
10:     $\rho_t \leftarrow \frac{\pi(A_t|S_t)}{\pi_b(A_t|S_t)}$.
11:     $e_t \leftarrow \phi_t + \rho_t \gamma_t \lambda_t e_{t-1}$.
12:     $\theta_{t+1} \leftarrow \theta_t + \alpha\left[\delta_t e_t - \gamma_{t+1}(1-\lambda_{t+1})(e_t^\top w_t)\bar{\phi}_{t+1}\right]$.
13:     $w_{t+1} \leftarrow w_t + \beta\left[\delta_t e_t - (\phi_t^\top w_t)\phi_t\right]$.
14: end for
```

Derivation To derive GQ(λ) with linear function approximation for GVFs, we follow the derivation of GQ(λ) for standard value functions in chapter (8). However, there are some subtle differences due to policy termination condition and varying eligibility traces. For the full derivation, I refer the reader to Appendix (A).

9.3 General state-value functions

So far, in this chapter, we have considered only action-value functions. Action-value function are important because they determine which primitive action to take from a given states in order to achieve a goal. However, state-value function are needed for various optimal control methods, such as policy gradient method with actor-critic architecture.

Analogous to action-value functions, general state-value functions are defined in the following form:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}[R_{t+1} + R_{t+2} + \cdots + R_{t+k} + Z_{t+k} \mid \pi, \gamma] \\ &\approx V_{\theta}(s) = \theta^{\top}\phi(s), \end{aligned} \tag{9.4}$$

where the true state-value, $V^{\pi}(s)$, satisfies a Bellman-equation (under the MDP condition), bringing the possibility of temporal-difference learning.

In addition to linear function approximation, we include eligibility traces and let the eligibility trace function $\lambda : \mathcal{S} \to [0,1]$ be an arbitrary function of state, thus varying over time.

To learn, modifiable parameters, θ, we extend GTD(λ) to the case where γ and λ functions depends on states. The complete derivation of GTD(λ), for this general setting, is shown in Appendix B.

Given all these, the algorithm is specified as follows: The weight vector θ is initialized arbitrarily. The secondary weight vector w is initialized to zero. An auxiliary memory vector known as the eligibility trace e is also initialized to zero. Their update rules are:

$$\theta_{t+1} = \theta_t + \alpha_t \left[\delta_t e_t - \gamma_{t+1}(1-\lambda_{t+1})(w_t^{\top} e_t)\phi_{t+1}\right], \tag{9.5a}$$

$$w_{t+1} = w_t + \beta_t \left[\delta_t e_t - (w_t^{\top}\phi_t)\phi_t\right], \tag{9.5b}$$

$$e_t = \rho_t\left(\phi_t + \gamma_t\lambda_t e_{t-1}\right), \tag{9.5c}$$

$$\delta_t = \left(R_{t+1} + (1-\gamma_{t+1})Z_{t+1} + \gamma_{t+1}\theta_t^{\top}\phi_{t+1}\right) - \theta_t^{\top}\phi_t, \tag{9.5d}$$

$$\rho_t = \frac{\pi(A_t \mid S_t)}{\pi_b(A_t \mid S_t)}, \tag{9.5e}$$

and $(\alpha_t, \beta_t; t \geq 0)$ are constant or decreasing step-size sequences (deterministic and positive). Table (5), shows how to use GTD(λ) for learning parameters of general state-value functions with linear function approximation.

Algorithm 5 GTD(λ) for GVFs

```
1: Initialize $w_0$ to 0, and $\theta_0$ arbitrarily.
2: Choose proper (small) positive values for $\alpha$, $\beta$.
3: Repeat for each episode:
4: Initialize $e = 0$
5: Take $A_t$ from $S_t$ according to $\pi_b$, and arrive at $S_{t+1}$.
6: Observe sample, $(\phi_t, R_{t+1}, Z_{t+1}, \phi_{t+1})$, where $\phi_t = \phi(S_t)$.
7: for each observed sample do
8:     $\lambda_t \leftarrow \lambda(S_t)$, $\gamma_t \leftarrow \gamma(S_t)$,$\gamma_{t+1} \leftarrow \gamma(S_{t+1})$.
9:     $\delta_t \leftarrow \big(R_{t+1} + (1-\gamma_{t+1})Z_{t+1} + \gamma_{t+1}\theta_t^\top \phi_{t+1}\big) - \theta_t^\top \phi_t$.
10:    $\rho_t \leftarrow \frac{\pi(A_t|S_t)}{\pi_b(A_t|S_t)}$.
11:    $e_t \leftarrow \rho_t\,(\phi_t + \gamma_t\lambda_t e_{t-1})$.
12:    $\theta_{t+1} \leftarrow \theta_t + \alpha\left[\delta_t e_t - \gamma_{t+1}(1-\lambda_{t+1})(e_t^\top w_t)\phi_{t+1}\right]$.
13:    $w_{t+1} \leftarrow w_t + \beta\left[\delta_t e_t - (\phi_t^\top w_t)\phi_t\right]$.
14: end for
```

9.4 Option conditional predictions with GQ(λ)

The theory of options in reinforcement learning, was first developed by Sutton, Precup, and Singh (1998, 1999). An option is a triple $o = (\mathcal{I}, \pi, \gamma)$, where $\mathcal{I} \subseteq \mathcal{S}$, $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is the policy followed during o. $\mathcal{I}$, normally is called initiation set, characterizes states in which are excited (active) for option policy π.

For example, consider a mobile robot, which is trying to dock to charge its battery. Here, $\mathcal{I}$ includes all states in which the charger is visible, π is a controller, which could be either hand-coded or learned, and robot will terminate docking option from state s with probability of $1 - \gamma(s)$ if the dock or charger is not in sight from state s.

Option conditional predictions are a form of temporally abstract predictions, which are essential as the means for representing abstract, higher-level knowledge about courses of action (or options). There are some benefits to this form of abstraction: (1) representing actions at a proper time scale can potentially help for faster learning and planning, (2) the problem can be transformed into several sub-tasks, and also allows domain knowledge be incorporate into the learning system.

The GQ(λ) algorithm is suitable for option conditional predictions. The algorithm allows the agent to learn about large number of options, that are excited at particular states, while the agent behaves according to its own policy.

9.5 Representing knowledge with GVFs

How can we know lots of things about how the world works, what can we do, and how can we apply them efficiently to maximize reward? We know so much about low-level sensory signals, how can we relate them to higher-level world knowledge? How can it all be organized and maintained? What are the principles?

In this chapter, we showed GQ(λ) is suitable for learning GVFs, which can be used as semantics for expressing knowledge (Sutton et al., 2011). GVFs are predictions and are always in the form of answers to the questions—in order to make a prediction, a question needs to be asked. Questions are like: “If I keep moving forward now, would I bump to a wall after few steps?”, or in the game of chess “What particular actions would lead to a win?”. Answer to these questions can be formulated in the form of a function that takes the current state as its input and gives a scalar *value function* as output. The questions address what to learn. The problem of prediction is how to learn value functions.

Based on these semantics, any answer to a question which is predictive is called knowledge. In other words, prediction is a way of expressing knowledge. This knowledge is obtained from sensory data communicated back and forth between agent and environment, and as such we call it grounded world knowledge, or in short, world knowledge (see Sutton et al., 2011, Maei and Sutton, 2010).

Of course, we already have made the working hypothesis that the world knowledge is predictive. Given all these, we can see that GQ(λ) is suitable for learning experientially grounded knowledge of the world. For example, Horde architecture (Sutton et al., 2011) was implemented in a robotic task, which uses GQ(λ) for learning in order to predict the world knowledge in a complex and nonstationary environment.

9.6 Conclusion

In this chapter we extended GQ(λ) and GTD(λ) to include general value functions (GVFs), which can be considered as a semantics for knowledge representation. We showed how to use the GQ(λ) and GTD(λ) algorithms to learn GVFs. In addition, our algorithms incorporate varying eligibility traces and policy termination function. These extensions makes these algorithms suitable for learning option-conditional predictions.

Chapter 10

Perspectives and Prospects

In this thesis, we presented a new family of temporal-difference learning algorithms. This is the first time that such algorithms have been developed while retaining their stability for general settings, including nonlinear function approximation and off-policy learning. Our gradient-TD algorithms can be viewed as performing stochastic gradient-descent in a mean-square projected Bellman-error (MSPBE) objective function whose optimum is the TD-solution. Another key and novel technique was introducing a set of auxiliary weights that were used in the update of actual parameters. The auxiliary weights were trained online, based on a stochastic update rule. All of our gradient-TD algorithms are guaranteed to converge.

Eligibility traces are essential to TD learning because they bridge the temporal gaps in cause and effect when experience is processed at a temporally fine resolution. Thus, we incorporated eligibility traces into our gradient-TD methods. Particularly, we considered a general scenario that eligibility trace function can be an arbitrary function of state, thus, can vary over time.

Another contribution is extending and making gradient-TD algorithms suitable for learning general value functions (GVFs). In standard RL, the value of a state is a measure of predicted future rewards from that state by following a policy. However, in the real-world, predictions are not necessarily in the form of future rewards. We would like to be able to answer questions such as: “If I keep moving forward now, would I bump to a wall after few second?” This question not only considers the outcome of moving forward after a few second, it also suggest that the policy of moving forward is only excited in particular states. The GVFs formulation is suitable for answering these temporally abstract predictions, which are also essential for representing abstract, higher level-knowledge about courses of action, or options. Here, we let policies to have their own activation and termination conditions, thus, making them suitable for option-conditional predictions.

Although, in this thesis, I focused on TD methods in the context of reinforcement learning, all these new ideas and methods can also be extended to various approximate DP methods. A good way to understand the impact of this work is to see it a a way of curing the curse-of-dimensionality.

The curse appears in large class of decision-making problems or problems that involve learning from interaction. Dynamic Programming (DP) is a general approach for solving complex decision-making problems. However, due to Bellman's curse-of-dimensionality it has not been extended in solving large-scale applications. The curse is considered as a malediction that has stopped scientists from finding exact solutions for large number of practical problems (Bellman, 1961, p. 94). Approximate DP methods—a collection of techniques for finding approximate DP solution— have been proposed to cure the curse.

However, approximate DP methods have cured the curse only partially. Baird's counterexample (cf. Chapter 2) suggests we may have a stability problem when using approximate DP methods. Particularly, the problem arises when we are interested in algorithms that have the following desirable features: incremental, online, and linear complexity both in terms of memory and per-time-step computation.

Although our work is just a starting point and has focused on TD methods, because TD algorithms are a class of approximate DP methods, our techniques and ideas can be extended to various approximated DP approaches. Thus, our work solves the problem of curse-of-dimensionality to a great extent.

We also have conducted empirical studies on comparing gradient-TD methods and conventional TD algorithms. Our empirical results suggest that gradient-TD methods' rate of convergence may be slower than conventional TD methods on on-policy problems where conventional TD methods sound. Our most recent gradient-TD methods are hybrid in that their asymptotic rate of convergence can be just as fast as TD methods on on-policy problems (see Appendix C).

10.1 Future Research Directions

Empirical results on hybrid gradient-TD methods Hybrid gradient-TD methods are promising methods to make gradient-TD methods more efficient. It turns out we can show hybrid methods have the same asymptotic rate of convergence of conventional TD methods on on-policy problems; that is, problems that conventional TD methods sound.

An important future direction would be to evaluate the effectiveness of hybrid gradient-TD algorithms on various on-policy and off-policy problems and demonstrate their empirical rate of convergence agains other gradient-TD methods as well as conventional TD algorithms.

Making gradient-TD methods more efficient In addition of hybrid gradient-TD idea, here I propose several potential ideas, which can make the gradient-TD algorithms more efficient in terms of rate of convergence.

The first natural method is proper choice of step-sizes. Particularly, we are interested in step-size methods that are not restricted to stationarity conditions; that is, are adaptive to new streams of data. One promising adaptive step-size method is *IDBD*, in which the step-sizes are adjusted based on gradient-descent on an objective error measure (Sutton, 1992). One of the problems with IDBD is that it has its own tuneable parameter, which is called the meta step-size parameter, and the performance of the main stochastic algorithm highly depends on the right choice of meta step-size parameter, which is varying for different problems. This means practitioners must spend a lot of effort to find the right value for meta step-size parameter. Mahmood (2010) suggests, Autostep, a free-parameter-tuning algorithm for step-size adaptation, which is robust and less sensitive to the choice of meta step-size parameter for large class of problems. Although the algorithm is proposed for supervised learning problems, it can also be extended to gradient-TD methods.

In addition, there is another motivation for using Autostep in gradient-TD algorithms. Gradient-TD methods require two step-size parameters–one for the main learning parameters and another one for auxiliary weights. This means, practitioners need to spend a lot of effort to tune the step-size parameters for particular problems. As the second set of weights have an underlying supervised learning update, Autostep can directly be applied to eliminate the second set of step-size parameter associated with auxiliary weights.

Another promising idea is to let the gradient TD-correction term, in TDC-like methods, have its own step-size parameter for the update. Remember that the main update is composed of two terms—the TD update and a gradient TD-correction term— and both have the same step-sizes. Of course, this will introduce a third step-size and thus another tuning parameter, however, Autostep, potentially can resolve the issues related to step-size tuning.

And finally, I think, regularizing the auxiliary weights can improve the rate of convergence. Note that the gradient TD-correction term is a function of auxiliary weights which eventually converges to zero. When the learning parameters are getting closer to the solution, the auxiliary weights get closer to zero and due to the noise, they can have a large effect on the direction of the update used in the main algorithm. Thus, regularizing the auxiliary weights (e.g. ridge regression) may potentially help us to reduce the noise and thus improve the rate of convergence.

Knowledge representation with GVFs and option conditional predictions In our everyday life a large amount of knowledge is in the form of temporally extended predictions which are goal oriented. General value functions (GVFs), to some great extent, can convey this notion of knowledge and as such can be used as a semantic for representing world-knowledge grounded in steam of experience.

Our desiderata for this form of knowledge, which is predictive and is grounded in received data (in the form of sensory signals received back and forth between the intelligence agent and the environment) should be useful for various kinds of things such as planning and autonomously be verifiable.

Thus, it would be interesting to apply our general algorithms, GQ(λ) or GTD(λ), on various temporal data, including robot data to learn about large amount of robot interactions with its environment and gather various kinds of predictive knowledge. GQ(λ), has been applied on robot data to learn about various kinds world-knowledge (Sutton, 2011), however, it has not been extended to learn option models (Sutton et al, 1998; 1999) and the Dyna architecture for planning purposes (Sutton, 1990)

We would like to have algorithms that can learn about multiple options at the same time—parallel learning— from a single stream of experience generated according to behavior policy. Thus, option conditional predictions involve off-policy learning, and gradient-TD methods allows options to be learned in conjunction with function approximation. Therefore, implementing gradient-TD methods on option conditional predictions and testing their efficiency is very beneficial.

Extensions to policy gradient methods In this thesis, I presented only the value-function based methods. Policy gradient methods are proposed as an alternative to value-function based methods in order to approximate the optimal policy (Sutton et al., 2000). In value-function based approaches, which we have discussed so far, the approximated value function has been used to select actions without constructing an explicit form for policy. This action selection procedure, which is based on value function, corresponds to computing a deterministic policy. However, policies are normally stochastic (Singh, Jaakkola, and Jordan, 1994) and, as a result, value-function based methods can cause some limitations. For example, for some problems (e.g., Gordon, 1995), convergence of the policy cannot be assured because an arbitrary small change in value function can cause one not to select some actions that would be selected by the optimal policy (Bertsekas and Tsitsiklis, 1996;

Gordon, 1995).

In policy gradient methods a stochastic policy is explicitly represented by a function approximation with its own parameters independent of the value function. The policy parameters at every time step are updated online according to the gradient of expected return (e.g., average reward) with respect to the policy parameters. For example, the policy might have a neural network structure whose input neurones are features representing state, whose synaptic weights are the policy parameters, and whose output neurones represent action selection probabilities.

The basic idea behind all policy gradient methods is to obtain an unbiased estimate for the gradient term by sample trajectories. These sample trajectories are informed by an estimate of average return, which is computed by conventional TD methods. Thus, with our new theoretical developments on TD methods, it is natural to implement them in places where TD has been used, such as policy gradient methods.

The Actor-critic algorithms are one of the most popular sub-class of policy gradient methods (Sutton et al., 2000; Peters et al., 2005, Bhatnagar et al., 2009). The Actor-critic methods also are considered a class of TD methods with two separate structure of actor and critic. The actor structure explicitly represents the action-selection policy and is independent of critic structure which estimates the value function. The critic addresses the prediction problem while the actor deals with the control problem. The actor and critic, simultaneously, are updated online based on TD ideas.

Thus future work will be to develop an effective actor-critic algorithm that is guaranteed to converge reliably under off-policy training and is suitable for large-scale real-world applications. Particularly, we seek a convergent policy gradient algorithm that: (1) Works with actor-critic architecture, (2) Works with temporal-difference learning, (3) Updates the policy parameters according to a novel policy gradient theorem, (4) Can learn off-policy and as such would allow using data effectively, (5) Has an incremental and online stochastic gradient update rule for both the actor and critic, and as such is scalable to large-problems.

10.2 Closing remarks

The new gradient-TD algorithms are solving two open problems in reinforcement learning: (1) Convergent $O(d)$ off-policy learning, (2) Nonlinear TD learning. We conclude that function approximation in reinforcement learning now is nearly as straightforward as supervised learning by using a gradient-descent method on a novel objective function (mean-square projected Bellman-error). This indicates that the curse-of-dimensionality is almost broken and general learning from interaction now is practical. The learning rate can still be improved by hybrid Gradient-TD algorithms (see Appendix C).

References

Antos, A., Munos, R., and Szepesvári, Cs. (2007). Fitted Q-iteration in continuous action-space MDPs. In *Advances in Neural Information Processing Systems 20*, pp. 916. MIT Press.

Antos, A., Szepesvári, Cs., Munos, R. (2008). Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning 71*:89–129.

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

Baird, L. C. (1999). Reinforcement Learning Through Gradient Descent. *PhD thesis*, Carnegie-Mellon University.

Barnard, E. (1993). Temporal-difference methods and Markov models. *IEEE Transactions on Systems, Man, and Cybernetics 23*(2):357–365.

Baxter, J., Tridgell, A., Weaver, L. (1998). Experiments in parameter learning using temporal differences. *International Computer Chess Association Journal*, 21, pp. 8499.

Bellman, R. (1961). *Adaptive control Processes*. Princeton University Press.

Benveniste, A., Metivier, M., Priouret, P. (1990). Adaptive Algorithms and Stochastic Approximations. Springer-Verlag.

Bertsekas, D. P., Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., Lee, M. (2009). Natural actor-critic algorithms. *Automatica* 45 (11): 2471-2482.

Borkar, V. S. (1997). Stochastic approximation with two timescales. *Systems and Control Letters 29*:291-294.

Borkar, V. S. (2008). Stochastic Approximation: A Dynamical Systems Viewpoint. Cambridge University Press.

Borkar, V. S. and Meyn, S. P. (2000). The ODE method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control And Optimization 38*(2):447–469.

Boyan, J. A. & Moore, A.W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, pp. 369–376. MIT Press.

Boyan, J. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning 49*:233–246.

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

Crites, R. H. & Barto, A.G. (1995). Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8*, pp. 1017-1023. MIT Press.

Delyon, B. (1996). General results on the convergence of stochastic algorithms. *IEEE Trans. Automat. Contr., vol. 41*, pp. 1245–1255.

Kushner, H. J. & Yin, G. G. (2003). Stochastic Approximation Algorithms and Applications. Second Edition, Springer-Verlag.

Geramifard, A., Bowling, M., Sutton, R. S. (2006). Incremental least-square temporal difference learning. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pp. 356–361.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 261268. Morgan Kaufmann.

Gordon, G. J. (2000). Reinforcement learning with function approximation converges to a region. In *Advances in Neural Information Processing Systems 13*, pp. 1040-1046. MIT Press.

Lagoudakis, M., Parr, R. (2003). Least squares policy iteration. *Journal of Machine Learning Research*,4:1107-1149.

Ljung, L. (1977). Analysis of recursive stochastic algorithms. *IEEE Tran. Automat. Control*, 22:551575, 1977.

Papavassiliou, V. A., and Russell, S. (1999). Convergence of reinforcement learning with general function approximators. In *Proceedings of the 16th international joint conference on Artificial intelligence*, pp. 748–757. Morgan Kaufmann.

Peters, J., Vijayakumar, S. and Schaal, S. (2005). Natural Actor-Critic. In *Proceedings of the 16th European Conference on Machine Learning*, pp. 280–291.

Maei, H. R. and Sutton, R. S. (2010). GQ(λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the Third Conference on Artificial General Intelligence*, pp. 91–96. Atlantis Press.

Maei, H. R., Szepesvari, Cs, Bhatnagar, S., Precup, D., Silver D., Sutton, R. S. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems 22*. MIT Press.

Maei, H. R., Szepesvari, Cs, Bhatnagar, S., Sutton, R. S. (2010). Toward off-policy learning control with function approximation. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 719–726. Omnipress.

Mahmood A. (2010). Automatic step-size adaptation in incremental supervised learning. *Master's Thesis*, University of Alberta.

Marbach, P., Mihatsch, O., Schulte, M., and Tsitsiklis, J. (1997). Reinforcement learning for call admission control and routing in integrated service networks. In *Advances in*

Neural Information Processing Systems 10. MIT Press.

Pearlmutter, B. A (1994). Fast exact multiplication by the hessian. *Neural Computation*, 6(1):147–160.

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

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

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

Schaeffer, J., Hlynka, M., Jussila, V. (2001). Temporal difference learning applied to a high-performance game-playing program. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 529534.

Scherrer, B. (2010). Should one compute the Temporal Difference fix point or minimize the Bellman Residual? The unified oblique projection view. In *In Proceedings of the 27th International Conference on Machine Learning*, pp. 959-966. Omnipress.

Silver, D., Sutton, R. S., Müller, M. (2007). Reinforcement learning of local shape in the game of Go. *Proceedings of the 20th International Joint Conferences on Artificial Intelligence*, pp. 1053–1058.

Silver, D. (2009). Reinforcement Learning and Simulation-Based Search in Computer Go. *Ph.D. thesis*. University of Alberta

Singh, S. P., Jaakkola, T., Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision problems. In *Proceedings of the 11th International Conference on Machine Learning*, pp. 284292.

Sutton, R. S. (1984). Temporal Credit Assignment in Reinforcement Learning. *Ph.D. Dissertation*. University of Massachusetts.

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

Sutton, R. S. (1992). Adapting bias by gradient descent: An incremental version of delta-bar-delta. *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 171–176, MIT Press.

Sutton, R. S. (1996). Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In *Advances in Neural Information Processing Systems 8*, pp. 1038-1044. MIT Press.

Sutton, R. S. (2009). The grand challenge of predictive empirical abstract knowledge. *Working Notes of the IJCAI-09 Workshop on Grand Challenges for Reasoning from Ex-*

periences.

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

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

Sutton, R. S., Precup D., and Singh, S. (1998). Intra-Option Learning about Temporally Abstract Actions. In *Proceedings of the 15th International Conference on Machine Learning*, pp. 556-564. Morgan Kaufmann.

Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *In Advances in Neural Information Processing Systems 12*, pp. 1057–1063. MIT Press.

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

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

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

Szepesvári, Cs., Smart, W. D. (2004). Interpolation-based Q-learning. In *Proceedings of the 21th International Conference on Machine Learning*, pp .791–798. ACM.

Szepesvári, Cs. (2010). Algorithms for Reinforcement Learning. Morgan Claypool Publishers.

Tesauro, G. J. (1992). Practical issues in temporal difference learning. *Machine Learning 8*: 257-277.

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

Tadić, V. (2001). On the convergence of temporal-difference learning with linear function approximation. In *Machine Learning*, 42:241267.

Tsitsiklis, J. N., and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674–690.

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

Appendix A

Derivation of GQ(λ) for GVFs and Varying Eligibility Traces

To derive GQ(λ) for general value functions, we follow GQ(λ) derivation steps in Chapter 8.

A.1 Forward-view objective function based on importance sampling

Define the following $\lambda-$weighted return based on importance sampling:

$$G_t^{\lambda\rho}(\theta) = R_{t+1} + (1-\gamma_{t+1})Z_{t+1} + \gamma_{t+1}\left[(1-\lambda_{t+1})\theta^\top\bar{\phi}_{t+1} + \lambda_{t+1}\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta)\right],$$

where

$$\bar{\phi}_t = \sum_a \pi(a \mid S_t)\phi(S_t, a),$$

$$\rho_t = \frac{\pi(A_t \mid S_t)}{\pi_b(A_t \mid S_t)},$$

and $G_t^{\lambda\rho}(\theta)$ is a noisy guess of future rewards of policy π, while the agent follows policy π. Let

$$\delta_t^{\lambda\rho}(\theta) \overset{\text{def}}{=} G_t^{\lambda\rho}(\theta) - \theta^\top\phi_t, \tag{A.1}$$

and

$$\delta_t^{\lambda}(\theta) \overset{\text{def}}{=} G_t^{\lambda}(\theta) - \theta^\top\phi_t, \tag{A.2}$$

where $G_t^{\lambda}(\theta)$ is $G_t^{\lambda\rho}(\theta)$ with $\rho = 1$.

The next theorem shows $\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t$ can be replaced by $\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]$.

Theorem 10. *(Off-policy TD with important weighting) Let π_b and π denote the behavior and target policies, respectively. Consider $\delta^\lambda(\theta)$, $\delta^{\lambda\rho}(\theta)$ defined in equations* (A.2) (A.1). *Then,*

$$\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t = \mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big], \tag{A.3}$$

where $\mathcal{P}_\mu^\pi$ is defined in Equation (7.4).

Proof. Let $\zeta_{t+1} = R_{t+1} + (1-\gamma_{t+1})Z_{t+1}$ and consider expanding the right-hand side of

$$\begin{aligned}
&\mathbb{E}\Big[G_t^{\lambda\rho}(\theta) \mid S_t = s, A_t = a\Big] \\
&= \mathbb{E}\Big[\zeta_{t+1} + \gamma_{t+1}\Big((1-\lambda_{t+1})\theta^\top\bar{\phi}_{t+1} + \lambda_{t+1}\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta)\Big) \mid S_t = s, A_t = a\Big] \\
&= \mathbb{E}\Big[\zeta_{t+1} + \gamma_{t+1}(1-\lambda_{t+1})\theta^\top\bar{\phi}_{t+1} \mid S_t = s, A_t = a, \pi\Big] \\
&\quad + \mathbb{E}\Big[\gamma_{t+1}\lambda_{t+1}\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta) \mid S_t = s, A_t = a\Big] \\
&= \mathbb{E}\Big[\zeta_{t+1} + \gamma_{t+1}(1-\lambda_{t+1})\theta^\top\bar{\phi}_{t+1} | S_t = s, A_t = a, \pi\Big] \\
&\quad + \sum_{s'} P(s' \mid s, a)\sum_{a'} \pi_b(a' \mid s')\frac{\pi(a'|s')}{\pi_b(a' \mid s')}\mathbb{E}\Big[\gamma_{t+1}\lambda_{t+1}G_{t+1}^{\lambda\rho}(\theta) \mid S_{t+1} = s', A_{t+1} = a'\Big] \\
&= \mathbb{E}\Big[\zeta_{t+1} + \gamma_{t+1}(1-\lambda_{t+1})\theta^\top\bar{\phi}_{t+1} \mid S_t = s, A_t = a, \pi\Big] \\
&\quad + \sum_{s',a'} P(s' \mid S_t, A_t)\pi(a' \mid s')\mathbb{E}\Big[\gamma_{t+1}\lambda_{t+1}G_{t+1}^{\lambda\rho} \mid S_{t+1} = s', A_{t+1} = a'\Big] \\
&= \mathbb{E}\Big[\zeta_{t+1} + \gamma_{t+1}(1-\lambda_{t+1})\theta^\top\bar{\phi}_{t+1} + \mathbb{E}\Big[\gamma_{t+1}\lambda_{t+1}G_{t+1}^{\lambda\rho}(\theta) \mid S_{t+1}\Big] \mid S_t = s, A_t = a, \pi\Big]
\end{aligned}$$

which, as it continues to roll out, gives us

$$\mathbb{E}\Big[G_t^{\lambda\rho}(\theta) \mid S_t = s, A_t = a\Big] = \mathbb{E}\Big[G_t^\lambda(\theta) \mid S_t = a, A_t = a, \pi\Big].$$

And, eventually it yields:

$$\mathbb{E}\Big[\delta^{\lambda\rho}(\theta)\phi_t\Big] = \mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t,$$

because the state-action distribution is based on behavior state-action pair distribution, μ.

□

Thus, the forward-view objective function, which is derived based on importance weighting, is as follows:

$$\begin{aligned} &J(\theta) \\ &= \left(\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t\right)^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1} \left(\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t\right) \\ &= \mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] \end{aligned} \tag{A.4}$$

A.1.1 Backward-view objective function

In this section, we transform TD forward-view, and as a result the proposed objective function, to mechanistic backward-view. To do this, we propose the following theorem, which provides a great tool for forward-view to backward-view transformation.

Theorem 11. *(Equivalence of TD forward-view and backward-view) The forward-view description of TD update is equivalence to the following mechanistic backward-view:*

$$\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi\Big] = \mathbb{E}[\delta_t(\theta)e_t], \tag{A.5}$$

where $\delta_t^{\lambda\rho}(\theta)$ is defined in Equation (A.1), *$\delta_t(\theta) = \zeta_{t+1} + \gamma_{t+1}\theta^\top\bar{\phi}_{t+1} - \theta^\top\phi_t$, and e_t is eligibility trace vector at time-step t, and has the following recursive update:*

$$e_t = \phi_t + \gamma\lambda\rho_t e_{t-1}. \tag{A.6}$$

Proof. Let $\zeta_{t+1} = R_{t+1} + (1-\gamma_{t+1})Z_{t+1}$ and consider

$$\begin{aligned} &G_t^{\lambda\rho}(\theta) \\ &= \zeta_{t+1} + \gamma_{t+1}\Big[(1-\lambda_{t+1})\theta^\top\bar{\phi}_{t+1} + \lambda_{t+1}\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta)\Big] \\ &= \Big[\zeta_{t+1} + \gamma_{t+1}(1-\lambda_{t+1})\theta^\top\bar{\phi}_{t+1}\Big] + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta) \\ &= \Big(\zeta_{t+1} + \gamma_{t+1}\theta^\top\bar{\phi}_{t+1}\Big) - \gamma_{t+1}\lambda_{t+1}\theta^\top\bar{\phi}_{t+1} + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta) \\ &= \Big(\zeta_{t+1} + \gamma_{t+1}\theta^\top\bar{\phi}_{t+1} - \theta^\top\phi_t + \theta^\top\phi_t\Big) - \gamma_{t+1}\lambda_{t+1}\theta^\top\bar{\phi}_{t+1} + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta) \\ &= \Big(\delta_t(\theta) + \theta^\top\phi_t\Big) - \gamma_{t+1}\lambda_{t+1}\theta^\top\bar{\phi}_{t+1} + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}G_{t+1}^{\lambda\rho}(\theta) \\ &\quad + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}\Big(\theta^\top\phi_{t+1} - \theta^\top\phi_{t+1}\Big) \\ &= \Big(\delta_t(\theta) + \theta^\top\phi_t\Big) + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}\Big(G_{t+1}^{\lambda\rho}(\theta) - \theta^\top\phi_{t+1}\Big) \\ &\quad + \gamma_{t+1}\lambda_{t+1}\Big(\rho_{t+1}\theta^\top\phi_{t+1} - \theta^\top\bar{\phi}_{t+1}\Big) \\ &= \Big(\delta_t(\theta) + \theta^\top\phi_t\Big) + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}\delta_t^{\lambda\rho}(\theta) + \gamma_{t+1}\lambda_{t+1}\Big(\rho_{t+1}\theta^\top\phi_{t+1} - \theta^\top\bar{\phi}_{t+1}\Big), \end{aligned}$$

thus,

$$\begin{aligned}
\delta_t^{\lambda\rho}(\theta) &= G_t^{\lambda\rho}(\theta) - \theta^\top \phi_t \\
&= \delta_t(\theta) + \gamma\lambda\rho_{t+1}\delta_t^{\lambda\rho}(\theta) + \gamma_{t+1}\lambda_{t+1}\left(\rho_{t+1}\theta^\top\phi_{t+1} - \theta^\top\bar{\phi}_{t+1}\right).
\end{aligned}$$

Note that the last part of the above equation has expected value of vector zero under the behavior policy because

$$\begin{aligned}
\mathbb{E}[\rho_t\phi_t \mid S_t] &= \sum_a \pi_b(a \mid S_t)\frac{\pi(a \mid S_t)}{\pi_b(a \mid S_t)}\phi(a \mid S_t) \\
&= \sum_a \pi(a \mid S_t)\phi(S_t, a) \equiv \bar{\phi}_t.
\end{aligned}$$

Putting all these together, we can write the TD update (in expectation) in a simple way in terms of eligibility traces which leads to backward-view:

$$\begin{aligned}
&\mathbb{E}\Big[\delta_t^{\lambda\rho}\phi_t\Big] \\
&= \mathbb{E}\Big[\Big(\delta_t + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}\delta_{t+1}^{\lambda\rho}\Big)\phi_t\Big] + \mathbb{E}\Big[\gamma_{t+1}\lambda_{t+1}\theta^\top\left(\rho_{t+1}\phi_{t+1} - \bar{\phi}_{t+1}\right)\phi_t\Big] \\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}\Big[\gamma_{t+1}\lambda_{t+1}\rho_{t+1}\delta_{t+1}^{\lambda\rho}\phi_t\Big] + 0 \\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}\Big[\gamma_t\lambda_t\rho_t\delta_t^{\lambda\rho}\phi_{t-1}\Big] \\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}_b[\gamma_t\lambda_t\rho_t\Big(\delta_t + \gamma\lambda\rho_{t+1}\delta_{t+1}^{\lambda\rho} + \gamma_{t+1}\lambda_{t+1}\theta\theta\left(\rho_{t+1}\phi_{t+1} - \bar{\phi}_{t+1}\right)\Big)\phi_{t-1}] \\
&= \mathbb{E}[\delta_t\phi_t] + \mathbb{E}[\gamma_t\lambda_t\rho_t\delta_t\phi_{t-1}] + \mathbb{E}\Big[\gamma_t\lambda_t\rho_t\gamma_{t+1}\lambda_{t+1}\rho_{t+1}\delta_{t+1}^{\lambda\rho}\phi_{t-1}\Big] + 0 \\
&= \mathbb{E}[\delta_t\left(\phi_t + \gamma_t\lambda_t\rho_t\phi_{t-1}\right)] + \mathbb{E}\Big[\gamma_{t-1}\lambda_{t-1}\rho_{t-1}\gamma_t\lambda_t\rho_t\delta_t^{\lambda\rho}\phi_{t-2}\Big] \\
&\vdots \\
&= \mathbb{E}_b\Big[\delta_t\Big(\phi_t + \gamma_t\lambda_t\rho_t\phi_{t-1} + \gamma_t\lambda_t\rho_t\gamma_{t-1}\lambda_{t-1}\rho_{t-1}\phi_{t-2} + \cdots\Big)\Big] \\
&= \mathbb{E}[\delta_t e_t], \qquad \text{(A.7)}
\end{aligned}$$

where $e_t = \phi_t + \gamma_t\lambda_t\rho_t e_{t-1}$, which gives us a backward view algorithm for the TD(λ) update.

□

A.1.2 Stochastic gradient-descent derivation

Now from Equation (A.4) and Theorem 11, we get

$$J(\theta) = \mathbb{E}[\delta_t(\theta)e_t]^\top \, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\,.$$

Following the derivation of GQ(λ), we get:

$$\begin{aligned} &-\frac{1}{2}\nabla J(\theta) \\ &= -\frac{1}{2}\nabla\Big(\mathbb{E}[\delta_t(\theta)e_t]^\top \, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\Big) \\ &= -\nabla\mathbb{E}\Big[\delta_t(\theta)e_t^\top\Big]\, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\ &= -\mathbb{E}\Big[\left(\gamma_{t+1}\bar{\phi}_{t+1} - \phi_t\right) e_t^\top\Big]\, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\,. \end{aligned} \tag{A.8}$$

We use the following identity:

$$\begin{aligned} \mathbb{E}\Big[\phi_t e_t^\top\Big] &= \mathbb{E}\Big[\phi_t(\phi_t + \gamma_t\lambda_t\rho_t e_{t-1})^\top\Big] \\ &= \mathbb{E}\Big[\phi_t\phi_t^\top + \gamma_t\lambda_t\rho_t\phi_t e_{t-1}^\top\Big] \\ &= \mathbb{E}\Big[\phi_t\phi_t^\top + \gamma_{t+1}\lambda_{t+1}\rho_{t+1}\phi_{t+1}e_t^\top\Big] \\ &= \mathbb{E}\Big[\phi_t\phi_t^\top + \gamma_{t+1}\lambda_{t+1}\bar{\phi}_{t+1}e_t^\top\Big]\,, \end{aligned}$$

where we have used $\mathbb{E}\big[\rho_{t+1}\phi_{t+1}e_t^\top\big] = \mathbb{E}\big[\bar{\phi}_{t+1}e_t^\top\big]$.
Thus, from above and Equation (A.8), we get

$$\begin{aligned} &-\frac{1}{2}\nabla J(\theta) \\ &= -\mathbb{E}\Big[\left(\gamma\bar{\phi}_{t+1} - \phi_t\right) e_t^\top\Big]\, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\ &= -\mathbb{E}\Big[\gamma_{t+1}\bar{\phi}_{t+1}e_t^\top - \phi_t e_t^\top\Big]\, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\ &= -\mathbb{E}\Big[\gamma_{t+1}\bar{\phi}_{t+1}e_t^\top - \Big(\phi_t\phi_t^\top + \gamma_{t+1}\lambda_{t+1}\bar{\phi}_{t+1}e_t^\top\Big)\Big]\, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\ &= -\mathbb{E}\Big[\gamma_{t+1}(1-\lambda_{t+1})\bar{\phi}_{t+1}e_t^\top - \phi_t\phi_t^\top\Big]\, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\ &= \mathbb{E}[\delta_t(\theta)e_t] - \mathbb{E}\Big[\gamma_{t+1}(1-\lambda_{t+1})\bar{\phi}_{t+1}e_t^\top\Big]\, \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\ &= \mathbb{E}[\delta_t(\theta)e_t] - \mathbb{E}\Big[\gamma_{t+1}(1-\lambda_{t+1})\bar{\phi}_{t+1}e_t^\top\Big]\, w(\theta), \end{aligned} \tag{A.9}$$

where $w(\theta) = \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]$.
Thus, by direct sampling from the above gradient-descent direction and weight-duplication trick we get:

$$\theta_{t+1} = \theta_t + \alpha_t\Big[\delta_t e_t - \gamma_{t+1}(1-\lambda_{t+1})\bar{\phi}_{t+1}(e_t^\top w_t)\Big],$$
$$w_{t+1} = w_t + \beta_t\left[\delta_t e_t - (w_t^\top \phi_t)\phi_t\right].$$

Appendix B

Derivation of GTD(λ) for General State-Value Functions and Varying Eligibility Traces

Note, many steps here follows the derivation steps in Chapter 7. Here, let us define the λ-return, for general value functions, according to

$$G_t^{\lambda\rho}(\theta) = \rho_t \left(R_{t+1} + (1-\gamma_{t+1})Z_{t+1} + \gamma_{t+1} \left[(1-\lambda_{t+1})\theta^\top \phi_{t+1} + \lambda_{t+1} G_{t+1}^{\lambda\rho}(\theta) \right] \right),$$

where

$$\rho_t = \frac{\pi(A_t \mid S_t)}{\pi_b(A_t \mid S_t)}.$$

Let

$$\delta_t^{\lambda\rho}(\theta) \quad = \quad G_t^{\lambda\rho}(\theta) - \theta^\top \phi_t. \tag{B.1}$$

Given the above definition, for the case where $\rho = 1$ (on-policy), we define

$$\delta_t^{\lambda}(\theta) \overset{\text{def}}{=} G_t^{\lambda}(\theta) - \theta^\top \phi_t, \tag{B.2}$$

and our usual one-step TD error is:

$$\delta_t(\theta) = R_{t+1} + (1-\gamma_{t+1})Z_{t+1} + \gamma_{t+1}\theta^\top \phi_{t+1} - \theta^\top \phi_t. \tag{B.3}$$

Here, the derivations are very similar to the derivation of GQ(λ) thus we avoid many similar steps.

Theorem 12. *(Off-policy TD for important weighting) Let π_b and π denote the behavior* *and target policies, respectively. Consider $\delta_t^{\lambda}(\theta)$, $\delta_t^{\lambda\rho}(\theta)$ defined in equations* (B.2) (B.1). *Then,*

$$\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t = \mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big], \tag{B.4}$$

where $\mathcal{P}_\mu^\pi$ is defined in Equation (7.4).

Proof. The proof is similar to forward-view/backward-view derivations for GQ(λ), which was derived in Appendix A. □

The forward-view objective function, thus, is as follows:

$$\begin{aligned} &J(\theta) \\ &= \left(\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t\right)^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1} \left(\mathcal{P}_\mu^\pi \delta_t^\lambda(\theta)\phi_t\right) \\ &= \mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big]^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi_t\Big] \end{aligned} \tag{B.5}$$

B.1 Backward-view objective function

The derivation in this section is also very similar to GQ derivations for mechanistic backward-view with subtle differences that we observed in derivation of GTD(λ) for standard value functions, thus we avoid details due to similarities.

Theorem 13. *(Equivalence of TD forward-view and backward-view) The forward-view description of TD update is equivalence to the following mechanistic backward-view:*

$$\mathbb{E}\Big[\delta_t^{\lambda\rho}(\theta)\phi\Big] = \mathbb{E}[\delta_t(\theta)e_t], \tag{B.6}$$

where $\delta_t^{\lambda\rho}(\theta)$ is defined in Equation (B.1), *$\delta_t(\theta)$ is the conventional TD error,*

$$\delta_t(\theta) = R_{t+1} + (1-\gamma_{t+1})Z_{t+1} + \gamma_{t+1}\theta^\top\phi_{t+1} - \theta^\top\phi_t,$$

and e_t is eligibility trace vector at time-step t, and has the following recursive update:

$$e_t = \rho_t\left(\phi_t + \gamma_t\lambda_t e_{t-1}\right). \tag{B.7}$$

B.2 Derivation of the GTD(λ) algorithm

Now from Equation (B.5) and Theorem 13, we get

$$J(\theta) = \mathbb{E}[\delta_t(\theta)e_t]^\top \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t].$$

We have:

$$
\begin{aligned}
&-\frac{1}{2}\nabla J(\theta) \\
&= -\frac{1}{2}\nabla\Big(\mathbb{E}[\delta_t(\theta)e_t]^\top\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\Big) \\
&= -\nabla\mathbb{E}\Big[\delta_t(\theta)e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= -\mathbb{E}\Big[(\gamma_{t+1}\phi_{t+1}-\phi_t)\,e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]\,.
\end{aligned}
$$

We use the following identity:

$$
\begin{aligned}
\mathbb{E}\Big[\phi_t e_t^\top\Big] &= \mathbb{E}\Big[\phi_t\rho_t(\phi_t+\gamma_t\lambda_t e_{t-1})^\top\Big] \\
&= \mathbb{E}\Big[\phi_t\rho_t\phi_t^\top+\phi_t\gamma_t\lambda_t e_{t-1}^\top\Big] \\
&= \mathbb{E}\Big[\phi_t\rho_t\phi_t^\top+\phi_{t+1}\rho_t\gamma\lambda e_t^\top\Big] \\
&= \mathbb{E}\Big[\phi_t\phi_t^\top+\phi_{t+1}\gamma_{t+1}\lambda_{t+1}e_t^\top\Big]\,,
\end{aligned}
$$

thus,

$$
\begin{aligned}
-\mathbb{E}\Big[(\gamma_{t+1}\phi_{t+1}-\phi_t)\,e_t^\top\Big] &= \mathbb{E}\Big[\gamma_{t+1}\phi_{t+1}e_t^\top-\phi_t e_t^\top\Big] \\
&= -\mathbb{E}\Big[\gamma_{t+1}\phi_{t+1}e_t^\top-\Big(\phi_t\phi_t^\top+\phi_{t+1}\gamma_{t+1}\lambda_{t+1}e_t^\top\Big)\Big] \\
&= \mathbb{E}\Big[\phi_t\phi_t^\top-\gamma_{t+1}(1-\lambda_{t+1})\phi_{t+1}e_t^\top\Big]\,.
\end{aligned}
$$

Using the above identity, we get

$$
\begin{aligned}
&-\frac{1}{2}\nabla J(\theta) \\
&= -\mathbb{E}\Big[(\gamma_{t+1}\phi_{t+1}-\phi_t)\,e^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= \mathbb{E}\Big[\phi_t\phi_t^\top-\gamma_{t+1}(1-\lambda_{t+1})\phi_{t+1}e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= \mathbb{E}[\delta_t(\theta)e_t]-\mathbb{E}\Big[\gamma_{t+1}(1-\lambda_{t+1})\phi_{t+1}e_t^\top\Big]\,\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t] \\
&= \mathbb{E}[\delta_t(\theta)e_t]-\mathbb{E}\Big[\gamma_{t+1}(1-\lambda_{t+1})\phi_{t+1}e_t^\top\Big]\,w(\theta), \qquad \text{(B.8)}
\end{aligned}
$$

where $w(\theta)=\mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_t(\theta)e_t]$, which yields the GTD($\lambda$) algorithm after direct sampling.

Appendix C

Hybrid Gradient-TD Methods

To derive hybrid gradient-TD methods, first consider derivation of GTD2/TDC in Chapter 5. The update equation for GTD2/TDC comes from (5.7) (or (5.9)). Let $A = \mathbb{E}\big[\phi(\phi - \gamma\phi')^\top\big]$ and $C = \mathbb{E}\big[\phi\phi^\top\big]$. Then, the expected update of TDC/GTD2, assuming we know the exact value of w, that is, $w(\theta) = C^{-1}(-A\theta + b)$, where $b = \mathbb{E}[R\phi]$, is

$$\Delta\theta_t = \alpha_t A^\top C^{-1} \left(-A\theta_t + b\right).$$

The above linear system is stable because $-A^\top C^{-1} A\theta_t$ negative definite[1]. Note $C = \mathbb{E}\big[\phi\phi^\top\big]$ is a positive definite matrix, and as a result $A^\top C^{-1} A$ is also positive definite matrix. When training is done over off-policy data, then A may not be positive stable matrix and thus TD(0) may diverge.

Now consider the $A_{\pi_b} = \mathbb{E}\big[\phi_t(\phi_t - \gamma\phi_{t+1})^\top\big]$, which is computed from on-policy data (t index refers to both time-step and t^{th} sample). Then, one can prove A_{π_b} is always positive definite matrix (see Bertsekans and Tsitsiklis, 1996). Thus, if we use this new matrix instead of C matrix, we get

$$\Delta\theta_t = \alpha_t A^\top A_{\pi_b}^{-1} \left(-A\theta_t + b\right), \tag{C.1}$$

and still the above update remains convergence because $-A^\top A_{\pi_b}^{-1} A$ is negative definite matrix. Because $A_{\pi_b}^\top$ is also positive definite matrix (note A_{π_b} is not symmetric matrix), thus, another way to get a convergent algorithm is to replace C with $A_{\pi_b}^\top$, that is,

$$\Delta\theta_t = \alpha_t A^\top A_{\pi_b}^{\top\,-1} \left(-A\theta_t + b\right), \tag{C.2}$$

which again $-A^\top A_{\pi_b}^{\top\,-1} A$ is negaitive definite. That is, we got another hybrid algorithm. This update is interesting because it tells us if the learning problem is on-policy, that is,

[1]The matrix $A \in \mathbb{R}^{d\times d}$ is positive definite if all the real-part eigenvalues of $A + A^\top$ are positive. In other words, for any non-zero $z \in \mathbb{R}^d$, we have $z^\top A z > 0$.

$A = A_{\pi_b}$, then we get an update equivalent to expected TD(0) update. Another observation is that the condition number of all of the above hybrid methods becomes the same as conventional TD(0) for on-policy problems, thus, suggesting the same (asymptotically) rate of convergence.

Now we turn into the derivation of hybrid gradient-TD algorithms. We would like to derive stochastic algorithms whose update is composed of TD update and a gradient-TD correction term (TDC-style). To do this, consider updates (C.1) and (C.2), and note state, s, has an underlying state distribution $\mu(s)$ according to the behavior policy π_b. Thus, we have $A = C - \mathbb{E}\Big[\gamma\phi{\phi'}^\top \mid \pi\Big]$ and $A_{\pi_b} = C - \mathbb{E}\big[\gamma\phi_t\phi_{t+1}^\top\big]$.

First, let's start with the right-hand side of (C.1):

$$\begin{aligned}
& A^\top A_{\pi_b}^{-1}(-A\theta + b) \\
&= \Big(A_{\pi_b} - A_{\pi_b} + A^\top\Big) A_{\pi_b}^{-1}(-A\theta + b) \\
&= (-A\theta + b) + \Big(-A_{\pi_b} + A^\top\Big) A_{\pi_b}^{-1}(-A\theta + b) \\
&= (-A\theta + b) + \Big(-A_{\pi_b} + A^\top\Big)\omega(\theta), \\
&= \mathbb{E}[\delta(\theta)\phi \mid \pi] - \gamma\left(\mathbb{E}\Big[\phi'\phi^\top \mid \pi\Big] - \mathbb{E}\Big[\phi_t\phi_{t+1}^\top\Big]\right)\omega(\theta),
\end{aligned} \tag{C.3}$$

where $\omega(\theta) = A_{\pi_b}^{-1}(-A\theta + b)$ and $\delta(\theta) = R + \gamma\theta^\top\phi' - \theta^\top\phi$. Now we can conduct direct sampling from the above update and use weight-duplication trick:

$$\theta_{t+1} = \theta_t + \alpha_t\left[\delta_t\phi_t - \gamma\left(\phi'\phi_t^\top\omega_t - \phi_t\phi_{t+1}^\top\omega_t\right)\right] \tag{C.4a}$$

$$\omega_{t+1} = \omega_t + \beta_t\left(\delta_t + \gamma\phi_{t+1}^\top\omega_t - \phi_t^\top\omega_t\right)\phi_t, \tag{C.4b}$$

where $\delta_t = R + \gamma{\phi'}^\top\theta_t - \phi_t^\top\theta_t$, and the second set of of weights have an update similar to TD(0) whose reward is δ_t. The stability of this algorithm can be guaranteed using the same ODE approach we used in this thesis because A_{π_b} matrix is positive definite.

The importance-weighting idea, also, can be used to learn about other policies. In this case, for state-value predictions, we have $\delta_t = R_{t+1} + \gamma\phi_{t+1}^\top\theta_t - \phi_t^\top\theta_t$ and TD update part is multiplied by likelihood ratio ρ_t while ϕ' in correction term will be replaced by $\rho_t\phi_{t+1}$. For action-value function predictions, we only need to replace ϕ' with $\sum_a \pi(a \mid S_{t+1})\phi(S_{t+1}, a)$.

Now consider another alternative for deriving hybrid methods from Equation (C.2):

$$\begin{aligned}
& A^\top A_{\pi_b}^{\top\,-1}(-A\theta + b) \\
&= \left(A_{\pi_b}^\top - A_{\pi_b}^\top + A^\top\right) A_{\pi_b}^{\top\,-1}(-A\theta + b) \\
&= (-A\theta + b) + \left(-A_{\pi_b}^\top + A^\top\right) A_{\pi_b}^{\top\,-1}(-A\theta + b) \\
&= (-A\theta + b) + \left(-A_{\pi_b}^\top + A^\top\right) \hat{\omega}(\theta), \\
&= \mathbb{E}[\delta(\theta)\phi \mid \pi] - \gamma \left(\mathbb{E}\Big[\phi'\phi^\top \mid \pi\Big] - \mathbb{E}\Big[\phi_{t+1}\phi_t^\top\Big]\right) \hat{\omega}(\theta), \qquad \text{(C.5)}
\end{aligned}$$

where $\hat{\omega}(\theta) = A_{\pi_b}^{\top\,-1}(-A\theta + b)$. Again, by direct sampling from the above update and using weight-duplication trick, we get:

$$\theta_{t+1} = \theta_t + \alpha_t \left[\delta_t\phi_t - \gamma\left(\phi' - \phi_{t+1}\right)\phi_t^\top\hat{\omega}_t\right] \qquad \text{(C.6a)}$$

$$\hat{\omega}_{t+1} = \hat{\omega}_t + \beta_t \left[\left(\delta_t - \phi_t^\top\hat{\omega}_t\right)\phi_t + \gamma(\phi_t^\top\hat{\omega}_t)\phi_{t+1}\right]. \qquad \text{(C.6b)}$$

The interesting feature of the above update is that for the on-policy case, where $\phi' = \phi_{t+1}$, the gradient TD-correction term automatically disappears and we get TD(0) update. Unlike Equation (C.4b), the update (C.6b) is harder to describe, but using the index-shifting trick, $\mathbb{E}\big[\phi_t\phi_{t+1}^\top\theta\big] = \mathbb{E}\big[\phi_{t-1}\phi_t^\top\theta\big]$, we can write it in the form of

$$\hat{\omega}_{t+1} = \hat{\omega}_t + \beta_t\left(\delta_t + \gamma\phi_{t-1}^\top\hat{\omega}_t - \phi_t^\top\hat{\omega}_t\right)\phi_t, \qquad \text{(C.7)}$$

which is like TD(0) update with backward target value $\phi_{t-1}^\top\hat{\omega}_t$. In other words, it means that the $\hat{\omega}$ update is trying to predict the sum of all the (discounted) TD-error up to the current time.