Incremental Off-policy Reinforcement Learning Algorithms

by

Ashique Mahmood

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

Doctor of Philosophy

in

Statistical Machine Learning

Department of Computing Science

University of Alberta

Abstract

Model-free off-policy temporal-difference (TD) algorithms form a powerful component of scalable predictive knowledge representation due to their ability to learn numerous counterfactual predictions in a computationally scalable manner. In this dissertation, we address and overcome two shortcomings of off-policy TD algorithms that preclude a widespread use in knowledge representation: they often produce high variance estimates and may become unstable.

The issue of high variance occurs when off-policy TD algorithms use importance sampling (IS), a standard approach to adjusting estimates when the sample and the target distributions are different. IS performs this adjustment by scaling the estimate by the likelihood ratio, also known as the *importance sampling ratio*. However, this ratio also often produces high variance in the estimates. We provide two different approaches to overcome this shortcoming.

The first approach relies on a well-known alternative to the ordinary form of IS, called the *Weighted Importance Sampling* (WIS), which is known to produce much less variance but has been under-utilized in reinforcement learning tasks. The main obstacle to using WIS in large-scale reinforcement learning tasks is that it is not known how to systematically extend a simple lookup-table-based estimator such as WIS to parametric function approximation with real-time computationally scalable updates. We overcome this obstacle by developing new techniques for deriving computationally efficient equivalent TD updates. This also allows us to simplify the derivations of some existing computationally efficient algorithms as well as produce new ones.

In the second approach, we directly address the main factor underlying the high variance issue: the use of the IS ratios. Ratios play a key role in producing high variance estimation as they may vary drastically from one sample to another. We show that the ratios can be eliminated from the updates by varying the amount of bootstrapping, a sophisticated technique for allowing a spectrum of multi-step TD algorithms. The core idea is to allow the bootstrapping parameter, which controls the amount of bootstrapping, vary and shrink

in a specific way. The elimination of ratios from the updates using this approach results directly in the reduction of variance compared to IS-based algorithms while still retaining the multi-step advantage of TD algorithms. We empirically evaluate the algorithms based on both approaches using several idealized experiments. We show that the WIS-based algorithms outperform existing off-policy learning algorithms often with an order of magnitude margin. On the other hand, the bootstrapping-based methods can retain more effectively the multi-step advantage compared to the IS-based methods despite occasionally shrinking the bootstrapping parameter.

The issue of instability with off-policy TD algorithms may appear when both function approximation and bootstrapping are used together, a desirable combination in reinforcement learning tasks. We discuss the key factors responsible for instability by analyzing the deterministic counterpart of TD updates. We illustrate that the on-policy TD algorithm may also be unstable if the updates are scaled or the bootstrapping parameter is set differently for different states but remains stable with a state-dependent discount factor. The TD algorithm can be made stable in all these cases by ensuring that a certain matrix involved in the deterministic TD update is always positive definite. We provide a systematic approach to ensuring positive definiteness of this matrix, which amounts to selectively emphasizing and de-emphasizing the updates of the stochastic counterpart in a particular manner. The resulting algorithm, which we call the *emphatic TD algorithm*, is stable, do not introduce extra tunable parameters and its additional memory or computational complexity is negligible.

Finally, we provide computationally efficient methods to perform a search over features for improving the representation of learning systems. These search methods enable continual discovery and maintenance of relevant and useful predictions that can be used as a building block of scalable predictive knowledge representations in long-lived systems.

Preface

Several chapters of this thesis are based on published papers written in collaboration with other researchers. In the rest of the chapters, the majority of the contributions were originally produced for this thesis. These are Chapter 2, 3, 4, and 9.

In all the papers I first-authored, I was responsible for composing the manuscript, and my co-authors assisted me to a great extent with editing. Chapter 5 and 6 are based on a published paper (Mahmood, Hasselt & Sutton 2014) that originated from our effort to solve a decades-old problem of combining weighted importance sampling with function approximation, first recognized by Precup et al. (2000). Sutton first brought the existence of this problem to my attention and led our effort to solve this problem. I developed the algorithms, wrote the theorem proofs, and conducted the experiments. I discussed extensively with Hasselt and Sutton during various stages of the development of the algorithms and the design of the experiments. Hasselt helped me with improving the proofs.

Chapter 7 is based on a published paper (Mahmood & Sutton 2015) which resulted from our effort to extend the above work to linear computational complexity. The core idea in this work, which we called *usage*, is nurtured by a decades-old algorithmic intuition of Sutton. I developed the algorithms, wrote the theorem proofs and conducted the experiments.

Chapter 8 is based on a paper in preparation (Mahmood, Yu & Sutton 2017). In this work, I discovered the core idea of the paper, which we called *action-dependent bootstrapping*. Yu assisted me extensively in developing the algorithm, correcting the derivations, and editing the content. Sutton led the joint effort of our group to expand our understanding of the tree-backup algorithm, which encouraged me to originate this work.

Chapter 9 and 10 are based on a published journal paper (Sutton, Mahmood & White 2016). In this work, Sutton developed the emphatic algorithm and composed the manuscript. I helped to formulate the expected update of the emphatic algorithm, which Sutton used to prove its stability. I developed and conducted the experiments. White assisted with editing, derivations, and proof. The negative result of Section 9.4 was first brought to our attention by Huizhen Yu through email communication. Section 9.4, 9.6 and 9.7 were originally produced for this thesis.

Chapter 11 and 12 are based on a published paper (Mahmood & Sutton 2013), where I developed the algorithms and conducted the experiments. Sutton assisted me in refining the algorithms and designing the experiments.

Dedicated To Liza

Acknowledgements

Due to the excellent support from my friends and colleagues, the days of my PhD studies have been the best I have ever spent. I wish to express my sincere gratitude to my supervisor for supporting me as a mentor and a friend throughout my graduate studies. I am fortunate to have spent so many days with him discussing the philosophy of AI, mind, and life. The world would greatly benefit from his careful thoughts on the future of AI and humanity.

I am deeply grateful to the members of my final and candidacy examining committees—Michael Bowling, Patrick Pilarski, Csaba Szepesvari, Remi Munos, Dale Schuurmans, Robert Holte, and Marek Reformat. The time I spent with them during the exams was among the most memorable and clarifying moments.

I cannot express enough gratitude to Csaba Szepesvari for teaching me some essential machine learning concepts and allowing me to check with him from time to time to query about theoretical puzzles. I am also deeply grateful to Simon Haykin for appreciating and encouraging my works from the very beginning of my PhD studies.

I owe many thanks to my closest collaborators Thomas Degris, Patrick Pilarski, Hado van Hasselt, and Huizhen Yu. Many intuitions used in this work can be traced back to our numerous meetings and email conversations. I am thankful for the support and friendship during my graduate studies from Hamid Reza Maei, Joseph Modayil, and Harm van Seijen.

I would like to thank Alberta Innovates Technology Futures, National Science and Engineering Research Council, Alberta Innovates Centre for Machine Learning, and the University of Alberta for funding my works throughout my PhD studies.

I am thankful to my family members for their unending support. And my deepest gratitude goes to my wife Farzana Liza Yasmin, without whom this thesis would not have come into existence. Liza is my greatest influence in pursuing a PhD in AI. This work is an outcome of our love and perseverance.

Contents

List of Figures

List of Tables

List of Notations

We use two kinds of variations in the notations using letters: 1) small vs. big letters, and 2) plain vs. boldfaced letters. Scalars are always in plain letters, vectors in boldfaced small letters, and matrices in boldfaced big letters. Among the scalars, random variables are denoted by big letters when greek letters are not used, and constant quantities are always denoted by small letters.

$\stackrel{\text{def}}{=}$	Equality by definition
$\mathbf{I}$	The identity matrix
$\mathbf{0}$	Vector or matrix of all zero elements
$\mathbf{1}$	Vector of all unit elements
$[\boldsymbol{\alpha}]_i$	ith element of vector $\boldsymbol{\alpha}$
$[\boldsymbol{\alpha}]_{sa}$	Element of vector $\boldsymbol{\alpha}$ corresponding to state s and action a
$[\mathbf{A}]_{i,:}$	ith row of matrix $\mathbf{A}$
$[\mathbf{A}]_{:,j}$	jth column of matrix $\mathbf{A}$
$[\mathbf{A}]_{i,j}$	Element of matrix $\mathbf{A}$ at row i and column j
$[\mathbf{A}]_{sa,s'a'}$	An element of matrix $\mathbf{A}$ with row corresponding to state s, and action a, and column corresponding to state s', and action a'
$[\mathbf{A}]_i$	ith diagonal element of matrix $\mathbf{A}$
$\boldsymbol{\alpha}^\top \mathbf{b}$	Dot product of two vectors: $\boldsymbol{\alpha}^\top \mathbf{b} = \sum_i [\boldsymbol{\alpha}]_i [\mathbf{b}]_i$
α	Scalar step-size parameter
$\mathrm{E}[\cdot]$	Expectation of a random variable
$\Pr\{\cdot\}$	Probability of an event
$\mathcal{S}$	Set of all states, considered to be discrete and finite
$\mathcal{A}$	Set of all actions, considered to be discrete and finite
$s, s', \bar{s}$	States
$a, a', \bar{a}, b$	Actions
t	Discrete time step
S_t	State at time t
A_t	Action at time t
R_t	Reward upon arrival at state S_t
π	A stationary, stochastic policy
$\pi(a\|s)$	Probability of taking action a in state s under policy π
γ	Discount factor parameter
$\gamma(\cdot)$	State-dependent discounting function, allowing variable degree of termination
γ_t	Shorthand for $\gamma(S_t)$
$T(t)$	time step of the first termination following t: $T(t) = \min\{k > t : \gamma_k = 0\}$
$v_\pi(s)$	Value of state s under policy π
λ	Bootstrapping factor parameter
$\lambda(\cdot)$	State-dependent or (state-) action-dependent bootstrapping function
λ_t	Shorthand for either $\lambda(S_t)$ or $\lambda(S_t, A_t)$

$\boldsymbol{\phi}(\cdot)$	Feature vector observed for a given state or state-action pair
$\boldsymbol{\phi}_t$	Feature vector observed for a given state or state-action pair at time t
$\boldsymbol{\Phi}$	The feature matrix with a feature vector per state in each row

Chapter 1

Introduction

An essential component of an intelligent agent is the retention of a vast amount of worldly knowledge that the agent can use to achieve its own goal. It is equally true for artificial agents as well as humans. In cricket games a batsman, whose goal can be thought of as scoring as many runs as possible, can benefit greatly by using the knowledge of the opponent bowlers, the strategies used by the opponent captains, and the pitch condition of different cricket grounds. A self-driving car needs to know about roads, pedestrian movements, traffic rules specific to a certain province, and the consequence of different courses of action in anomalous situations in order to drive safely toward its destination. When building an artificial agent, designers typically hard-code such domain knowledge, which requires human intervention from time to time in order to perform recalibration and maintenance.

Predictions form a grounded and testable knowledge representation. If the knowledge of an artificial agent is represented in a predictive form, the agent can autonomously test and maintain it without requiring any human intervention. Such knowledge can be in terms of predictions of the agent's sensorimotor observations, conditional on different ways of behaving. To a self-driving car, a muddy road may simply mean that if it drives along the road, it may experience more bumps and less friction on the wheels through its sensation. Such predictive knowledge is inherently grounded in actions and sensations of the agent. It is also maintainable by the agent itself as it can test its knowledge by following a certain policy and see if the outcome matches its prediction.

An agent can use one of many machine learning algorithms to learn predictions from observations. There are batch learning algorithms such as Monte Carlo averages or least-squares methods that are known for their statistical efficiency. To learn a plethora of predictions in a scalable way, there are also learning algorithms that updates incrementally and, thus, are computationally more frugal. Moreover, due to the complexity of the state space of the real world, it is practical to parameterize the solution and use methods based on parametric function approximation.

Among machine learning algorithms for learning predictions, a powerful class called *off-policy learning algorithms* stands out for the ability of the algorithms to learn about

policy-contingent predictions without ever needing to follow the policy in question. This way the agent can expand enormously the amount of knowledge it can gather. A self-driving car can learn about the consequence of different ways of driving such as speeding or changing lanes frequently while in reality the car is following a relatively safer policy. Such conditional predictions are not only grounded in raw observations, but they also provide the ability to answer a multitude of *what-if* questions. In human psychology, such counterfactual thinking is thought to be of paramount importance for effective planning and decision making (Roese 1997, Baird & Fugelsang 2004).

In this dissertation, we consider the problem of learning off-policy predictions in a computationally scalable manner. A natural remedy to this problem is to resort to incremental off-policy learning algorithms with function approximation. In recent years, several incremental learning algorithms have been developed based on Temporal Difference (TD) learning. However, they face two issues: 1) these algorithms tend to have high variance, and 2) when combined with function approximation, they may cause instability. These preclude the use of incremental off-policy TD algorithms in large-scale practical applications. This dissertation explores these two shortcomings of off-policy TD algorithms and provides different solutions to overcome them.

1.1 Key Ideas and Approaches

We adopt the general value function (GVF) framework (Sutton et al. 2011) for representing off-policy prediction tasks. Under this framework, prediction tasks are represented as GVFs, which are a generalization of value functions with arbitrary state-dependent discounting. This expands policy evaluation tasks to a more general specification of predictions with conditional discounting at arbitrary time horizon. GVFs represent prediction tasks, often known as *predictive questions*, in a form that can be learned by one of various learning algorithms. The guesses produced by a learner are often called *predictive answers* or, in short, predictions.

Several ideas played key roles in our analyses and solutions to the high variance and instability issues in off-policy TD. They include importance sampling techniques, computationally efficient equivalent updates, and the deterministic counterpart of stochastic updates.

At the heart of off-policy prediction algorithms is a technique known as importance sampling (Hammersley & Handscomb 1964, Rubinstein 1981). This technique allows using samples drawn from a distribution different from the distribution under consideration. In off-policy prediction, importance sampling is used to learn about one policy while following another policy. The core importance sampling methods have origins in batch learning. The importance sampling technique relies on scaling the estimates with the likelihood ratio between the distributions, also known as *the importance sampling ratio*. As ratios may vary drastically from samples to samples, they result in high estimation variance and are also

the key source of the high variance issue of off-policy TD algorithms. Our approaches to resolving the high variance issue in off-policy TD revolve around understanding different importance sampling techniques, adopting superior methods from batch learning, and dispelling an explicit presence of the ratios while keeping the benefits of importance sampling.

Although the core idea behind off-policy prediction is rooted in batch learning, we eventually want algorithms that work online in a computationally efficient, that is, in an incremental manner. Of course, we can apply the algorithms for batch learning directly to online learning, but they will be computationally expensive and not suitable for continual use. This gap between batch learning algorithms and incremental algorithms can be bridged by deriving computationally efficient implementations of the batch learning algorithms for online learning while retaining equivalence of the parameters being computed. Such equivalence techniques have played a key role in this dissertation. We have developed powerful techniques for producing equivalent incremental updates from batch learning algorithms. These techniques help us adopt batch learning techniques for importance sampling and produce their equivalent incremental counterparts as well as understand the existing equivalent incremental updates better.

Although TD updates are inherently stochastic, they can be analyzed in terms of their deterministic counterparts. More specifically, for a given stochastic update, there exists a corresponding expected update with respect to the stationary distribution, and the stability of stochastic updates is associated with the convergence of the expected updates. This simplifies vastly the analysis of stability in TD algorithms, and consequently, we adopt it in our approach. By analyzing the expected updates of the TD algorithm, we look at the heart of the instability issue, its origin, different cases where instability occurs, and develop remedies.

1.2 Related Works

Since the introduction of incremental off-policy prediction algorithms (Precup et al. 2000), several works acknowledged the importance of adopting off-policy policy evaluation tasks for representing long-term predictions of sensorimotor events and have put this to work (Sutton et al. 1999, Sutton et al. 2006, Rafols 2006). The idea of representing long-term predictions in terms of value functions goes back to as far as the original TD paper (Sutton 1988). More recently, this idea is formalized by Sutton et al. (2011) under a general framework for value functions, which we call the *general value function (GVF) framework*. White (2015) comprehensively explored this framework.

The high variance issue of ordinary importance sampling is well documented in the Monte Carlo simulation literature (Kleijnen 1978, Andradóttir 1995, Liu 2001, Robert & Casella 2004). An alternative to ordinary importance sampling takes a form of weighted average and is called *weighted importance sampling*. This method is known for its superior theoretical properties, such as reduced variance in a large class of problems (Liu 2001) and

overall bounded variance (Precup et al. 2001). Moreover, weighted importance sampling often produces superior empirical performance compared to ordinary importance sampling (Hesterberg 1988, Casella & Robert 1998, Precup, Sutton & Singh 2000, Shelton 2001, Liu 2001, Koller & Friedman 2009, Paduraru 2013). However, this method is a tabular batch estimator, lacking a computationally efficient implementation or an extension to parametric function approximation. For this reason, weighted importance sampling has not been utilized in incremental off-policy TD algorithms. The necessity and the difficulty of utilizing weighted importance sampling in incremental off-policy TD with function approximation have been acknowledged more than a decade ago (Precup et al. 2001).

One of the first incremental off-policy TD algorithms for predictions was based on ordinary importance sampling introduced by Precup, Sutton, and Singh (2000). An extension of this method to linear function approximation was proposed by Precup, Sutton and Dasgupta (2001), but the high variance issue in this algorithm was readily acknowledged. To reduce variance, Precup et al. (2005) proposed a class of policies for which off-policy estimation produces minimum variance. This limits the set of allowable behavior policies, hence cannot be used for off-policy estimation with arbitrary policies.

The phenomenon of algorithmic equivalences between batch and incremental algorithms has firm roots in reinforcement learning. TD(λ) is one of the first temporal-difference based incremental algorithms for which an equivalence to a batch algorithm was established (Sutton 1988). For several classical incremental algorithms in reinforcement learning, such as Sarsa(λ) and different variants of Q(λ), such algorithmic equivalences have been analyzed (Sutton & Barto 1998). Recently, a new TD(λ) algorithm has been developed that achieves equivalence with an online algorithm with superior properties than the original TD(λ) (van Seijen & Sutton 2014). However, such equivalences have not been analyzed for incremental off-policy algorithms that use importance sampling. Moreover, the phenomenon of algorithmic equivalences for reinforcement learning itself has not been well explored. As a consequence, no systematic techniques for achieving such algorithmic equivalences have been developed, and this phenomenon is not well understood.

The instability issue has been demonstrated in a variety of domains and through simple counterexamples in several works (Bertsekas 1994, Baird 1995, Gordon 1995, Boyan & Moore 1995, Tsitsiklis & Van Roy 1996). The instability is shown to occur both under off-policy training (Baird 1995) and selective updating (Tsitsiklis & Van Roy 1996). Baird proposed to avoid this issue by considering a gradient-descent algorithm based on a different objective function than TD (Baird 1995). Later, a gradient-based approach was also developed for TD updates (Sutton et al. 2009, Maei & Sutton 2010, Maei 2011). However, these gradient-based TD algorithms are not fully satisfactory as they require the tuning of an extra step-size parameter and are also susceptible to the problem of high variance (Defazio & Graepel 2014).

Expected updates played a significant role in the analysis of TD algorithms in many

works. Sutton (1988) showed the stability of on-policy TD(λ) by analyzing its expected update. It is closely related to the ODE corresponding to stochastic processes and has been widely utilized in the analysis of stochastic approximation methods. However, the source of the instability issue for TD updates is not fully understood in terms of the expected updates. Expected updates also did not play a major role in the derivation of TD algorithms.

1.3 Contributions

This dissertation contains a number of contributions toward two different issues of off-policy learning algorithms, each overcoming a major obstacle to a widespread massive application of off-policy algorithms for scalable predictive knowledge representation: the high variance issue of off-policy algorithms and the instability issue of off-policy algorithms. We group these contributions according to these two obstacles and list them below:

1. Our contributions toward the issue of high variance can be divided further into two different approaches: those based on the technique of weighted importance sampling, and those based on bootstrapping-based ideas. We discuss these contributions in the following:

   - **WIS2:** Our first contribution based on weighted importance sampling is *WIS2*, a tabular estimator that is an improvement over the standard weighted importance sampling estimator. This estimator forms the basis for more generally applicable off-policy algorithms we developed in this thesis. WIS2 is introduced in Chapter 4.
   - **WIS-LS** & **WIS2-LS:** Our second contribution based on weighted importance sampling is developing a principled approach to extend the tabular weighted importance sampling estimator to the parametric function approximation. By applying this technique to the standard weighted importance sampling, we obtain WIS-LS algorithm, which can be applied to tasks beyond off-policy learning, such as, general supervised learning tasks where importance sampling is required to adjust the conditional distribution of the output. By applying this technique to WIS2, we obtain WIS2-LS algorithm, which we adopt for further extensions. These algorithms are introduced in Chapter 5.
   - **WIS-LSTD(λ):** represents the culmination of our contributions based on weighted importance sampling. This algorithm is the fullest extension of the weighted importance sampling technique into parametric function approximation with real-time updates, and a generalization of WIS2 and WIS2-LS. This contribution is described in Chapter 6.
   - **WIS-TD(λ), WIS-GTD(λ), WIS-TO-GTD(λ):** Our final contribution based on weighted importance sampling is a class of algorithms that emerged from our

efforts to incorporate the benefits of weighted importance sampling with updates of linear computational complexity. These algorithms are developed in Chapter 7.

- **ABQ(ζ):** Our second approach toward overcoming the high variance issue results to an off-policy algorithm, ABQ(ζ), that varies the amount of bootstrapping in an action-dependent manner to avoid an explicit use importance sampling ratios. By avoiding an explicit presence of the importance sampling ratios, this algorithm effectively reduces the estimation variance. This algorithm is introduced in Chapter 8.
- **The action-dependent bootstrapping framework:** A major contribution based on our second approach is to utilize action-dependent bootstrapping as a framework for analyzing some of the existing off-policy algorithms as well as producing new ones. This framework is developed in Chapter 8.

2. Our main contributions toward the issue of instability are listed below:

   - **Understanding the instability of TD(λ):** We presented a simplified approach for analyzing the instability of TD updates based on the properties of the corresponding deterministic updates. This leads to some new understanding of the instability issue involved with TD(λ). This analysis approach is described in Chapter 9.
   - **Emphatic TD:** Our most important contribution toward the instability issue of off-policy algorithms is the development of a new stable off-policy algorithm, we call the *emphatic TD algorithm*. In addition, we also provide a systematic way of developing stable off-policy algorithms. These contributions are described in Chapter 10.

In addition to these directed efforts, we have also made an additional contribution toward the understand of incremental learning algorithms. Although not part of the main issues we address in this thesis, this contribution played a pivotal role in developing a number of algorithms in this work. We describe it below:

3. **Algorithmic equivalence technique**: We developed a technique for systematically developing computationally frugal, incremental algorithms from non-incremental ones while keeping their outcomes equivalent. We used this algorithmic equivalence technique to develop WIS-LSTD(λ), WIS-TD(λ), WIS-GTD(λ), WIS-TO-GTD(λ), and ABQ(ζ). We illustrate this phenomenon of algorithmic equivalence through examples in Chapter 3 and introduce our most sophisticated technique in Chapter 6.

Finally, we return to our original motivation behind developing incremental off-policy algorithms, that is, to construct scalable predictive knowledge representation. We address

the problem of automatic discovery and retention of units of predictive knowledge representation as a search of feature representation. This may facilitate the agent for curating and retaining useful off-policy predictions as units of predictive knowledge representation.

4. Our main contributions toward addressing the feature discovering and retention problem are listed below:

    - **Representation search through generate and test**: Our first major contribution is to present the problem of feature discovery and retention as a representation search problem and introduce a series of incremental and computationally frugal search algorithms that discover and retain features effectively through the process of generate and test. This contribution is given in Chapter 11.
    - **Complimenting representation learning**: Our second major contribution here is to show that our representation search methods can be used complementarily with existing representation learning approaches, boosting their performance. This contribution is described in Chapter 12.

We describe the general value function framework in Chapter 2, which can formally specify long-term policy-contingent predictions.

Chapter 2

The Problem Setup and Background

In this chapter we define formally the off-policy prediction task. Primarily, we formulate the problem of predicting long-term sensorimotor events as off-policy policy evaluation tasks under the general value function framework. Then we describe different categories of task settings, depending on whether estimation is performed completely or approximately, leading to tabular and function approximation settings, or how samples are presented to and estimates are expected from the learner, leading to off-line, per-trajectory, and real-time learning settings. Real-time learning with function approximation is the most natural setting for large-scale prediction. However, it is important to discuss the other settings because learning algorithms often originate and are well understood in those idealized settings. Finally, we discuss different computational goals of learning algorithms for these settings.

2.1 The Agent-Environment Interaction Model

At the heart of all reinforcement learning problems is the agent-environment interaction model. In this interaction model, it is assumed that an agent interacts with the environment through actions, and in response, the environment changes its state and provides a reward. Figure 2.1 depicts the simplest model of agent-environment interaction. In this model, the agent has direct access to the state of the environment. This interaction is usually formulated using the mathematical framework of Markov Decision Processes (MDPs). Typically, this model is appended or modified based on the particular problem of interest. For example, the model is often modified by assuming that the agent does not directly access the states but rather receives observations or features.

More formally, let $\mathcal{S}$ denote the finite state space of the environment and $\mathcal{A}$ the finite action space. At each time step t, the agent is at state $S_t \in \mathcal{S}$ and chooses an action $A_t \in \mathcal{A}$ according to a fixed policy $\mu : \mathcal{A} \times \mathcal{S} \to [0, 1]$, where $\mu(a|s) \stackrel{\text{def}}{=\!=} \Pr\{A_t = a|S_t = s\}$ under policy μ. Upon taking the action, the agent transitions to the next state $S_{t+1} \in \mathcal{S}$ according to probability $p(s'|s, a) \stackrel{\text{def}}{=\!=} \Pr\{S_{t+1} = s'|S_t = s, A_t = a\}$ and receives a scalar

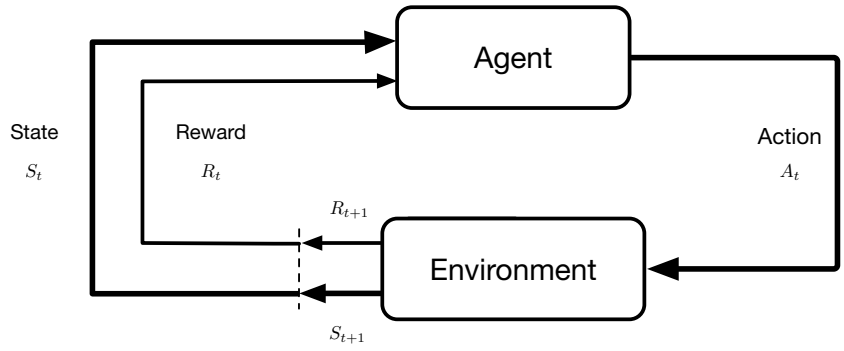

Figure 2.1: The agent-environment interaction model where an agent exerts actions and in response the environment changes states and provides a reward signal.

reward $R_{t+1} \in \mathbb{R}$. As the agent behaves according to policy μ, we call it the *behavior* policy.

2.2 Predictions as General Value Functions (GVFs)

Sutton et al. (2011, also see White 2015) introduced the general value function (GVF) framework to frame predictive questions using the well-established notion of value functions. As the name suggests, GVFs are a generalization of the value function concept in RL. They provide a powerful framework for open-ended prediction tasks by utilizing the well-established foundation of value functions and making value estimation algorithms available for learning predictions. Note that a GVF is not a solution method but rather is a specification of a predictive question. In the following, we provide the formal description of the GVF framework, while illustrating how predictions can be represented as GVFs by working with an example.

For an example of a predictive question, we consider that a robot is attempting to leave a room without running into a wall too many times. The robot has a particular routine for exiting the room that is a stationary policy and guarantees that the number of time steps to leave the room is finite. She is not trying to come up with the optimal way of leaving but rather is interested to know the consequence of following its current policy. More precisely, the robot is interested to know how many times she will run into a wall before leaving the room. We are assuming that the robot has sensors to determine whether she has run into a wall or has succeeded leaving the room.

This predictive question can be formally represented by the random predictive event associated with this question. This will also allow us to determine the true answer to this question so that we can verify the accuracy of any tentative answer. The predictive event involved here can be described in terms of a random variable that accounts for the number of times the robot runs into a wall, a random variable that determines the termination of the event, and the description of the policy the robot is following. All of these affect the

true answer to the predictive question. The true answer may also differ depending on where the robot is located in the room. For example, the robot will likely collide with the wall less number of times if she is located in front of the doorway as opposed to far away from it. Therefore, the predictive event depends on the initial situation.

The GFV framework provides the elements required for defining this predictive question. It also utilizes the MDP framework and uses states to allow situation-based prediction. A predictive event in GVFs is defined cumulatively in terms of a special signal, which is also known as a *pseudo-reward*. At each time step t, the agent, as it transitions from state S_t to S_{t+1}, receives a pseudo-reward R_{t+1} with mean $r(s,a,s') \stackrel{\text{def}}{=\!=} \mathrm{E}\left[R_{t+1}|S_{t+1}=s', A_t=a, S_t=s\right]$ and bounded variance. This is a generalization of rewards, and we use these terms interchangeably in this work.

For our predictive question, this pseudo-reward R_{t+1} can be the sensor signal that determines whether the robot has collided with a wall. Each time she collides with a wall, the signal value is $+1$, and zero otherwise. Note that in a control task, a collision would not be typically encoded as a positive reward, as maximizing it would result in more collisions. This is where GVFs diverges from the conventional value function framework and allows a more powerful formulation.

Besides the pseudo-reward signal, there are two other components of GVFs: a state-dependent discounting function $\gamma(s) : \mathcal{S} \to [0,1]$ determining the termination of the predictive event and the policy, upon which the prediction is contingent. The prediction event may terminate with probability $1-\gamma(s)$ upon arrival at each state s. Alternatively, $\gamma(s)$ can be thought as the degree of at s. The prediction event terminates with certainty when $\gamma(s)=0$. We use the shorthand $\gamma_t \stackrel{\text{def}}{=\!=} \gamma(S_t)$. Let L_t^h denote a trajectory of state, action, and pseudo-reward sequence starting with S_t and ending with S_h: $L_t^h \stackrel{\text{def}}{=\!=} \{S_t, A_t, R_{t+1}, S_{t+1}, \cdots, R_h, S_h\}$. Here, $h > t$, that is, trajectories must contain at least a single transition. Let us define $T(t)$ to be the time step of the first full termination after time step t, that is, $T(t) = \min\{k > t : \gamma_k = 0\}$. Then a *complete trajectory* of a predictive event starting at t is denoted as $L_t \stackrel{\text{def}}{=\!=} L_t^{T(t)}$. We denote the probability of continuation of a trajectory L_t^h as γ_{t+1}^h, where $\gamma_t^h \stackrel{\text{def}}{=\!=} \Pi_{k=t}^h \gamma_k$.

The prediction is contingent upon the agent following a fixed policy $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$, which can be different from the behavior policy μ. We call π the *target* policy. We use the shorthands $\pi_t \stackrel{\text{def}}{=\!=} \pi(A_t|S_t)$ and $\mu_t \stackrel{\text{def}}{=\!=} \mu(A_t|S_t)$. We assume that predictive events terminate almost surely: $\gamma_{t+1}^{\infty} = 0$ w.p.1, $\forall t$. A special case is where, for some states s, a full termination always occurs: $\gamma(s) = 0$. This is similar to episodic settings, where interactions occur in episodes, and at the end of the episode the agent goes back to and restarts from a state according to a fixed initial-state distribution. Unlike the episodic setting, the agent in our setting does not restart after arriving at a terminal state. It is only the predictive event that terminates. The agent continues to interact with the environment seamlessly according to the behavior policy.

For our predictive question, the state-dependent discounting function $\gamma(\cdot)$ can be based on the sensor signal that determines whether the robot succeeds leaving the room. For the first visited state s where the robot determines she has exited the room, the discount factor is $\gamma(s) = 0$, and it is 1 for all other states. The policy π is simply the routine the robot uses to exit the room. But it could also be some other stationary policy if the robot was curious about it. In that case, the robot would be asking the same predictive question, how many times she runs into a wall, but this time contingent on a policy different than what she is currently following, forming a counterfactual predictive question.

The predictive event can be given by a random variable defined as the discounted cumulative sum of the pseudo-rewards:

$$G_t = \sum_{l=t+1}^{T(t)} \prod_{i=t+1}^{l-1} \gamma_i R_l, \tag{2.1}$$

which corresponds to the trajectory L_t. This quantity is known as the *return*. The pseudo-rewards are also known as cumulants of the return. The true answer to the predictive question is then given by a general value function, denoted $v(s; \pi, r, \gamma)$ for each state s, and defined as:

$$v(s; \pi, r, \gamma) \stackrel{\text{def}}{=\!=} \mathrm{E}_\pi \left[G_t | S_t = s \right], \tag{2.2}$$

where $\mathrm{E}_\pi [\cdot|\cdot]$ stands for the expectation with respect to the probability distribution of L_t under policy π given that the trajectory starts from a particular state. In the conventional framework, this is known as the *state-value function*, as it determines how valuable each state is. As in action-value functions, which determines the value of each state-action pair, GVFs may also be formulated in terms of state-action pairs.

For our predictive question, the return G_t stands for the number of times the robot collided with a wall before leaving the room. Note that as soon as the robot leaves the room, all the cumulants from that time are fully discounted. Therefore, no collisions after the exit are taken into account in G_t. Moreover, by defining it in terms of states, the true prediction is allowed to be different based on where the robot is initially located.

The agent can learn predictions by estimating the general value function $v(s; \pi, r, \gamma)$ for a certain state s using data gathered from its own experience. This task in the conventional value function framework is known as the *policy evaluation* or the *state-value estimation task*. The experience of the agent is generated by following the behavior policy μ, which we say to be "off" the target policy π. In a learning task where there is a discrepancy between the behavior and the target policy, the problem is called the *off-policy* problem. In this case, the agent is learning policy-contingent predictions by estimating GVFs off-policy. When the target policy is the same as the behavior policy, the problem is known as the *on-policy* problem.

2.3 Model-free and Model-based Learning

A state-value estimation task can be further distinguished based on whether the agent is assumed to have access to the transition model of the environment. If such a model is known, it can be utilized in combination with data gathered from experience. Dynamic programming methods only rely on transition models and do not use data at all (Sutton & Barto 1998). Some other methods may utilize both to estimate state values (Paduraru 2013).

Methods that use a transition model in estimation are known as *model-based methods*. The assumption of having access to the environment's transition dynamics is unreasonable for practical and large-scale problems. To apply model-based methods in these problems, one requires to estimate the model separately from data before estimating state values. Model estimation is typically a computationally intensive process, and storing the model typically requires much more memory than that required for storing the state-value estimates. Model-based methods are also subject to model-approximation error.

Methods that do not explicitly maintain the model of the environment are known as *model-free methods*. These methods typically require much less memory and computation compared to model-based methods. However, model-free methods tend to use much more samples than model-based methods. These methods are often preferred as opposed to model-based methods due to their simplicity and computational congeniality, which is more pertinent to large-scale prediction we consider here.

2.4 Tabular and Linear Function Approximation Settings

One of the simplest learning settings for the value estimation problem is where the agent observes the states. This is often known as the *lookup-table* or *tabular setting* because the values can be stored in a table and looked up by the index of the states.

The tabular setting is contrasted with the *function approximation setting*. In this setting, we assume that the agent does not observe the states themselves but rather have access to a m-dimensional feature vector $\boldsymbol{\phi}(s) \in \mathbb{R}^m$ corresponding to each state s. We use the notational shorthand $\boldsymbol{\phi}_k \stackrel{\text{def}}{=\!=} \boldsymbol{\phi}(S_k)$. The goal of the learner in the function approximation setting is to approximate the value of a state s as a linear function of the features: $\boldsymbol{\theta}^\top \boldsymbol{\phi}(s) \approx v(s; \pi, r, \gamma)$ by learning the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^m$.

The function approximation setting specializes to the tabular setting when the feature vector for a state is a unique standard basis vector. If the feature vectors are standard basis vectors but not unique, they constitute what is often known as the *state-aggregation setting*.

2.5 Off-line, Per-trajectory, and Real-time Learning Settings

Often an algorithm is motivated in a simple idealized learning setting, but we seek to apply that algorithm to a more complicated real-life setting. Sometimes an algorithm efficient in one learning setting can become expensive in another one. In this section, we describe different reinforcement learning settings that differ by how the samples are presented to the learner. We will use these settings in Chapter 3 for illustrative examples of algorithmic equivalences.

A conceptually simple learning setting for state-value estimation with linear function approximation is where data samples are available as pairs of trajectories and the feature vectors corresponding to the initial state of the trajectories: $(L_t, \boldsymbol{\phi}_t)$. It is conceptually simple because the goal here is to estimate expected returns and the samples are available in terms of complete trajectories, from which returns can be easily computed. This setting is also pertinent in conventional tasks. In an episodic task, these trajectories can be collected after each episode completes. In a continuing task, these trajectories can be collected by waiting for a long period and computing the returns up to a certain precision.

This setting can be further distinguished depending on the way the learner is expected to produce an estimate. In supervised learning problems, often an estimate from the learner is expected only after a complete data set is presented or data becomes available to the learner only as a complete set. Once the learner is trained on the data set, the learned estimate is then used to produce predictions on unseen samples without further learning. We call such settings *off-line learning settings*.

Definition 1 (The off-line learning setting)**.** *A learning setting is called* off-line *if there is a clear separation in time between learning and the use of the learned estimate in performance, that is, the estimate is not learned further when being used.*

An example of off-line learning setting is where a set of n samples $\{(L_{t_k}, \boldsymbol{\phi}_{t_k})\}_{k=1}^{n}$ is given to the learner at once, where t_k denotes the initial time step of the kth trajectory. The learner is expected to produce a single estimate of the parameter vector $\boldsymbol{\theta}$. Once this estimate is produced, it is kept fixed and used to predict returns as $\boldsymbol{\theta}^\top \boldsymbol{\phi}_t$ for a given feature vector $\boldsymbol{\phi}_t$.

In contrast, some other settings have learning and the use of the learned estimates occurring simultaneously. We call these settings *on-line learning settings*.

Definition 2 (The online learning setting)**.** *A learning setting is called* online *if learning and the use of the learned estimates occur simultaneously, that is, learning does not stop when the estimate is being used.*

As this definition does not precisely determine how estimation and the presentation of samples for use are interleaved, this allows various forms of online learning settings, for example, where samples arrive as mini-batches or learned estimates are expected only once

in a while. An example of an online learning setting is where the learner is expected to produce a learned estimate after each sample $(L_t, \boldsymbol{\phi}_t)$ is received. This does not require an estimate every time step. As we can see, an on-line learning setting is closely related to how a sample is defined. In the current example, learning occurs on-line on a trajectory-by-trajectory basis. In the following, we will refer to this setting as the *per-trajectory learning setting*.

We discuss one final reinforcement learning setting, where samples arrive in real-time as the transitions occur. Hence, samples arrive as a tuple of feature vectors, actions, next rewards, and next feature vectors $(\boldsymbol{\phi}_t, A_t, R_{t+1}, \boldsymbol{\phi}_{t+1})$ each time a transition occurs from time step t to $t+1$. In this case, we are interested in making a learned estimate after each transition. For example, after the first transition, the sample $(\boldsymbol{\phi}_0, A_0, R_1, \boldsymbol{\phi}_1)$ becomes available, and the learner is expected to produce a learned estimate $\boldsymbol{\theta}_1$. Here learning occurs not only on-line but in real-time as transitions occur from one state to another. We call this the *real-time learning setting*.

We discussed different variations of reinforcement learning settings based on how samples are presented to the learner resulting in three settings: off-line learning, per-trajectory learning, and real-time learning settings. RL algorithms are typically tied to specific learning settings. For example, Monte Carlo methods are typically considered in either the off-line or the per-trajectory learning setting, whereas temporal-difference methods are considered in the real-time learning setting. It is possible to bring an algorithm well-suited for one learning setting and adopt it for another in some cases, allowing us to carry forward the benefits of the former algorithm to the latter setting.

2.6 Computational Goals for Learning Algorithms

When applying an algorithm from one setting into another, the question of memory and computational complexity becomes relevant. In this section, we explore some natural computational goals for learning algorithms when applied to a particular setting.

Different learning settings impose different computational goals for learning algorithms. In an off-line learning setting, the goal is typically to use as much computational resource as possible to get the most out of the data. As training data sets in practical applications are becoming larger and larger, there is also a pressing need to reduce the computational burden of off-line learning algorithms. However, sample complexity or data efficiency is more important and computational goals are more flexible in this setting. As long as computation completes in a reasonable amount of time requiring a feasible amount of memory, the computational burden of an algorithm is typically deemed acceptable.

In an on-line learning setting, computational goals play a bigger role. Algorithms with certain computational goals are more fitting to this setting than others. As data samples arrive one at a time, and a learned estimate is expected before the next sample is presented, it sets a natural restriction to how much computational resources the learner can use to learn

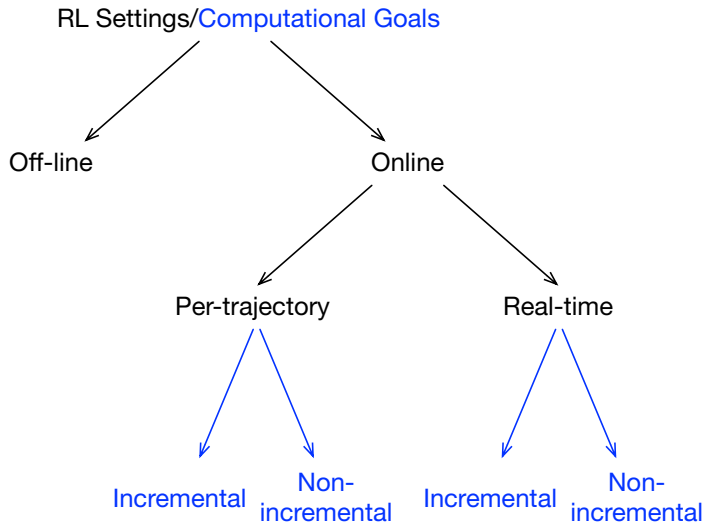

Figure 2.2: Reinforcement learning problems are categorized in three different settings: off-line, per-trajectory, and real-time (shown in black), based on how samples are presented to the learner. Two different computational goals of learning algorithms are incremental and non-incremental (shown in blue), and their precise definitions depend on computational and memory requirements as well as the adopted setting. For example, an algorithm that is incremental in a per-trajectory learning setting may not be incremental in a real-time learning setting.

and produce the estimate. If samples are arriving continually at a fixed and frequent rate, often the case in a real-time learning setting, then the learner has a fixed computational budget between two samples. This computation has to be expended on two computational elements: the *learning element* updating the estimate, and the *performance element* producing the prediction. This restriction rules out algorithms that require more and more computation per-sample as the number of samples increases. The online learning setting has implications on the memory requirement as well. Algorithms that require more and more memory as the number of samples increases are infeasible to use. An algorithm with slowly increasing memory complexity might be workable for a long period, but eventually has to be intervened.

The most fitting algorithms in such settings would not require more memory and per-sample computation as the number of samples increases. Sutton and Whitehead (1993) coined the term *strict incrementality* for this computational goal.

Definition 3 (Strictly-incremental algorithms)**.** *An algorithm is called* strictly incremental *if it does not require more memory or per-sample computation as the number of samples increases.*

Throughout this thesis, we focus on achieving algorithms that are strictly incremental. Algorithms that do not fulfill this computational requirement is called non strictly-

incremental algorithm. In the following, we use *incremental* interchangeably with strictly incremental.

Incrementality is more relevant in the on-line learning setting and depends on the definition of samples. For example, an algorithm that is incremental in a per-trajectory learning setting may not remain incremental in a real-time learning setting. It is because in the former setting samples arrive as complete trajectories, whereas in the latter setting samples arrive with transitions. In Figure 2.2, we illustrate these different categories of settings and computational goals of learning.

2.7 Bias, Consistency and Mean Squared Error of Average and Ratio Estimators

Almost all the estimators we discuss in this chapter are either in the form of a simple average:

$$\bar{X}_n = \frac{\sum_{k=1}^{n} X_k}{n}, \tag{2.3}$$

or in the form of a ratio of two simple averages:

$$\frac{\bar{X}_n}{\bar{Y}_n} = \frac{\sum_{k=1}^{n} X_k/n}{\sum_{k=1}^{n} Y_k/n} = \frac{\sum_{k=1}^{n} X_k}{\sum_{k=1}^{n} Y_k}. \tag{2.4}$$

Here, both $\{X_k\}_{k=1}^n$ and $\{Y_k\}_{k=1}^n$ are sequences of i.i.d. random variables, and Y_k is a dependent variable of X_k such that $\mathrm{E}[Y_1] = 1$. Both the average and the ratio are estimating $\mu_x = \mathrm{E}[X_1]$. We call the former estimator the *average estimator* and the latter the *ratio estimator*.

Two most commonly analyzed properties of estimators are their biases and consistencies. We analyze these two properties for all the estimators we discuss in this chapter.

Bias of an Estimator

A bias of an estimator V with respect to an unknown parameter v is defined as the quantity: $\mathrm{E}[V] - v$. Based on this definition, an unbiased estimator can be defined as follows.

Definition 4 (Unbiased estimator)**.** *Given a constant scalar $v \in \mathbb{R}$, an estimator $V \in \mathbb{R}$ is called an unbiased estimator of v if and only if*

$$\mathrm{E}[V] = v.$$

In the following, we describe some theoretical properties of these two estimators in terms of their biases.

Lemma 1 (Unbiasedness of average estimator)**.** *Given an i.i.d. sequence* $\{X_k\}_{k=1}^n$*, the average estimator defined by 2.3 is an unbiased estimator of* μ_x*.*

Proof. By taking the expectation of the average estimator, we get

$$\mathrm{E}[\bar{X}_n] = \mathrm{E}\left[\frac{\sum_{k=1}^n X_k}{n}\right] = \frac{\sum_{k=1}^n \mathrm{E}[X_k]}{n} = \frac{\sum_{k=1}^n \mathrm{E}[X_1]}{n} = \mathrm{E}[X_1]\frac{n}{n} = \mu_x. \tag{2.5}$$

□

The ratio estimator is generally a biased estimator of μ_x. However, under fairly general conditions, it can be shown that the bias of the ratio estimator goes down exponentially with n to zero. For that, we can use the first-order Taylor expansion of the bias. David and Sukhatme (1974) achieved this result by assuming that the values Y_k takes are bounded uniformly above zero. Eltinge (1994) weakened the condition by allowing a nontrivial proportion of the values to be zero. Their conditions also required the existence of higher moments. The following derivation provides an intuition on how the first order Taylor approximation of the bias is used to achieve this result:

$$\begin{aligned}
&\mathrm{E}\left[\frac{\bar{X}_n}{\bar{Y}_n}\right] - \mu_x = \mathrm{E}\left[\frac{\bar{X}_n - \mu_x\bar{Y}_n}{\bar{Y}_n}\right] && (2.6)\\
&= \mathrm{E}\left[\left(\bar{X}_n - \mu_x\bar{Y}_n\right)\left(\bar{Y}_n\right)^{-1}\right] && (2.7)\\
&= \mathrm{E}\left[\left(\bar{X}_n - \mu_x\bar{Y}_n\right)\left(1 + (\bar{Y}_n - 1)\right)^{-1}\right] && (2.8)\\
&= \mathrm{E}\left[\left(\bar{X}_n - \mu_x\bar{Y}_n\right)\left(1 + \sum_{k=1}^{\infty}(-1)^k(\bar{Y}_n - 1)^k\right)\right] && (2.9)\\
&\approx \underbrace{\mathrm{E}\left[\bar{X}_n - \mu_x\bar{Y}_n\right]}_{0} - \mathrm{E}\left[(\bar{X}_n - \mu_x\bar{Y}_n)(\bar{Y}_n - 1)\right] && (2.10)\\
&= -\mathrm{E}\left[(\bar{X}_n - \mu_x - \mu_x\bar{Y}_n + \mu_x)(\bar{Y}_n - 1)\right] && (2.11)\\
&= \mu_x\mathrm{E}\left[(\bar{Y}_n - 1)^2\right] - \mathrm{E}\left[(\bar{X}_n - \mu_x)(\bar{Y}_n - 1)\right] && (2.12)\\
&= \frac{1}{n}\left(\mu_x \operatorname{Var}[Y_1] - \operatorname{Cov}[X_1, Y_1]\right). && (2.13)
\end{aligned}$$

Thus the bias of the ratio estimator is $O\left(\frac{1}{n}\right)$.

Consistency of an Estimator

The definition of a consistent estimator depends on the definition of almost sure convergence, which we provide first.

Definition 5 (Almost sure convergence)**.** *A sequence of random variables* $\{X_k\}_{k=1}^n$ *is said to converge to another random variable* X *almost surely if and only if*

$$\Pr\left\{\lim_{n\to\infty} X_n = X\right\} = 1.$$

Almost sure convergence of the sequence $\{X_k\}_{k=1}^n$ is denoted in short by $X_n \xrightarrow{\text{a.s.}} X$.

Definition 6 (Consistent estimator)**.** *Given a constant scalar $v \in \mathbb{R}$, a sequence of real-valued estimators $\{V_k\}_{k=1}^n$ is called a consistent estimator of v if and only if $V_k \xrightarrow{\text{a.s.}} v$.*

The following Lemma known as the strong law of large numbers will be useful for proving consistency properties of the average and the ratio estimators.

Lemma 2 (Almost sure convergence of average estimator)**.** *Given an i.i.d. sequence of random variables $\{X_k\}_{k=1}^n$, the average estimator $\bar{X}_n$ defined by (2.3) converges almost surely to μ_x, that is, $\bar{X}_n \xrightarrow{\text{a.s.}} \mu_x$.*

Lemma 3 (Consistency of average estimator)**.** *Given an i.i.d. sequence of random variables $\{X_k\}_{k=1}^n$, the average estimator $\bar{X}_n$ defined by (2.3) is a consistent estimator of μ_x.*

Proof. It follows directly from Lemma 2 and Definitions 6. □

Lemma 4 (Consistency of ratio estimator)**.** *Given two i.i.d. sequences of random variables $\{X_k\}_{k=1}^n$ and $\{Y_k\}_{k=1}^n$ and $\mathrm{E}[Y_1] = 1$, the ratio estimator $\frac{\bar{X}_n}{\bar{Y}_n}$ defined by (2.4) is a consistent estimator of μ_x.*

Proof. By applying Lemma 3 to $\bar{X}_n$, we have $\bar{X}_n = \frac{\sum_{k=1}^n X_k}{n} \xrightarrow{\text{a.s.}} \mu_x$, and by applying to $\bar{Y}_n$, we have $\bar{Y}_n = \frac{\sum_{k=1}^n Y_k}{n} \xrightarrow{\text{a.s.}} \mathrm{E}[Y_1] = 1$, respectively. Then it follows that $\frac{\bar{X}_n}{\bar{Y}_n} \xrightarrow{\text{a.s.}} \frac{\mu_x}{\mathrm{E}[Y_1]} = \mu_x$, and hence the ratio estimator is a consistent estimator of μ_x. □

Mean Squared Error of an Estimator

The Mean Squared Error of an estimator V with respect to an unknown parameter v is defined to be the quantity: $\mathrm{E}[(V - v)^2]$. MSE can be written in terms of the bias and the variance of the estimator in the following way:

$$\begin{aligned}
\mathrm{E}[(V - v)^2] &= \mathrm{E}[(V - \mathrm{E}[V] + \mathrm{E}[V] - v)^2] && (2.14)\\
&= \mathrm{E}[(V - E[V])^2] + 2\underbrace{\mathrm{E}[(V - \mathrm{E}[V])(\mathrm{E}[V] - v)]}_{0} + \mathrm{E}[(\mathrm{E}[V] - v)^2] && (2.15)\\
&= \underbrace{\mathrm{E}[(V - E[V])^2]}_{\mathrm{Var}[V]} + \underbrace{(\mathrm{E}[V] - v)^2}_{\mathrm{Bias}^2}. && (2.16)
\end{aligned}$$

As the average estimator is an unbiased estimator, its MSE is simply its variance:

$$\mathrm{E}[(\bar{X}_n - \mu_x)^2] = \mathrm{Var}[\bar{X}_n] = \mathrm{Var}\left[\frac{\sum_{k=1}^n X_k}{n}\right] = \frac{1}{n}\mathrm{Var}[X_1]. \quad (2.17)$$

It is clear from the above that MSE of the average estimator reduces to zero as $n \to \infty$.

Just like its bias, the calculation of the MSE of the ratio estimator is tricky. Using similar conditions as was used for the approximation of bias, Eltinge (1994) showed that MSE of the ratio estimator reduces exponentially with n. The MSE of the ratio estimator can also be related to the MSE of the average estimator using the first order Taylor approximation of the variance (Liu 2001). The first-order Taylor approximation of the variance of the ratio estimator can be written as follows:

$$\operatorname{Var}\left[\frac{\bar{X}_n}{\bar{Y}_n}\right] = \mathrm{E}\left[\left(\frac{\bar{X}_n}{\bar{Y}_n} - \mathrm{E}\left[\frac{\bar{X}_n}{\bar{Y}_n}\right]\right)^2\right] \tag{2.18}$$

$$= \mathrm{E}\left[\left(\bar{X}_n - \bar{Y}_n \mathrm{E}\left[\frac{\bar{X}_n}{\bar{Y}_n}\right]\right)^2 \left(\bar{Y}_n\right)^{-2}\right] \tag{2.19}$$

$$= \mathrm{E}\left[\left(\bar{X}_n - \mu_x \bar{Y}_n\right)^2 \left(1 + (\bar{Y}_n - 1)\right)^{-2}\right] \tag{2.20}$$

$$\approx \mathrm{E}\left[\left(\bar{X}_n - \mu_x \bar{Y}_n\right)^2\right] \tag{2.21}$$

$$= \mathrm{E}\left[\bar{X}_n^2\right] + \mu_x^2 \mathrm{E}\left[\bar{Y}_n^2\right] - 2\mu_x \mathrm{E}[\bar{X}_n \bar{Y}_n] - 2\mu_x^2 + 2\mu_x^2 \tag{2.22}$$

$$= \mathrm{E}\left[\bar{X}_n^2\right] - \mu_x^2 + \mu_x^2 \mathrm{E}\left[\bar{Y}_n^2\right] - \mu_x^2 - 2\mu_x \mathrm{E}[(\bar{X}_n - \mu_x)\bar{Y}_n] \tag{2.23}$$

$$= \mathrm{E}\left[\bar{X}_n^2\right] - \mu_x^2 + \mu_x^2 \left(\mathrm{E}\left[\bar{Y}_n^2\right] - 1\right) - 2\mu_x \mathrm{E}[(\bar{X}_n - \mu_x)(\bar{Y}_n - 1)] \tag{2.24}$$

$$= \operatorname{Var}[\bar{X}_n] + \mu_x^2 \operatorname{Var}[\bar{Y}_n] - 2\mu_x \operatorname{Cov}[\bar{X}_n, \bar{Y}_n] \tag{2.25}$$

$$= \frac{1}{n}\left(\operatorname{Var}[X_1] + \mu_x^2 \operatorname{Var}[Y_1] - 2\mu_x \operatorname{Cov}[X_1, Y_1]\right). \tag{2.26}$$

Approximations are applied to remove the $O(n^{-2})$ terms. The MSE is simply proportional to the variance, because the squared of the bias is $O(n^{-2})$. Then the MSE of the ratio estimator can be related to the MSE of the average estimator in the following way:

$$\operatorname{MSE}\left[\frac{\bar{X}_n}{\bar{Y}_n}\right] \approx \operatorname{Var}\left[\frac{\bar{X}_n}{\bar{Y}_n}\right] \tag{2.27}$$

$$\approx \frac{1}{n}\left(\operatorname{Var}[X_1] + \mu_x^2 \operatorname{Var}[Y_1] - 2\mu_x \operatorname{Cov}[X_1, Y_1]\right) \tag{2.28}$$

$$= \operatorname{MSE}[\bar{X}_n] + \frac{1}{n}\left(\mu_x^2 \operatorname{Var}[Y_1] - 2\mu_x \operatorname{Cov}[X_1, Y_1]\right). \tag{2.29}$$

We denote the estimated MSE in the last expression by $\operatorname{EMSE}[\bar{X}_n]$. Using this estimated MSE, we can characterize when the leading terms in the MSE of the ratio estimator is smaller than the MSE of the average estimator. The condition that the estimated MSE of the ratio estimator is less than the MSE of the average estimator can be found in the following way:

$$\operatorname{EMSE}\left[\frac{\bar{X}_n}{\bar{Y}_n}\right] - \operatorname{MSE}[\bar{X}_n] < 0 \tag{2.30}$$

$$\implies \frac{1}{n}\left(\mu_x^2 \operatorname{Var}[Y_1] - 2\mu_x \operatorname{Cov}[X_1, Y_1]\right) < 0 \tag{2.31}$$

$$\implies \mu_x \operatorname{Cov}[X_1, Y_1] > \frac{\mu_x^2 \operatorname{Var}[Y_1]}{2}. \tag{2.32}$$

Therefore, the condition is that X_1 and Y_1 are highly positively correlated if $\mu_x > 0$. Similarly, if $\mu_x < 0$, X_1 and Y_1 need to be highly negatively correlated.

2.8 The Importance Sampling Technique

Importance sampling is a technique for re-adjusting samples to make them appear to be drawn according to one distribution while they are actually drawn according to another distribution. This re-adjustment is achieved by scaling samples by the likelihood ratio, which is also known as the *importance sampling ratio*. We use the terms the importance sampling ratio and the importance weight interchangeably.

For a trajectory L_t^h, the importance weight W_t^h given the start state S_t is defined as

$$W_t^h \stackrel{\text{def}}{=\!=} \frac{\prod_{k=t}^{h-1} \pi(A_k|S_k)p(S_{k+1}|S_k,A_k)}{\prod_{k=t}^{h-1} \mu(A_k|S_k)p(S_{k+1}|S_k,A_k)} = \frac{\prod_{k=t}^{h-1} \pi(A_k|S_k)}{\prod_{k=t}^{h-1} \mu(A_k|S_k)} = \prod_{k=t}^{h-1} \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}. \tag{2.33}$$

We define $W_t^t \stackrel{\text{def}}{=\!=} 1$. Note that the transition probabilities cancel in the numerator and the denominator. Therefore, we do not need to know the model of the environment in order to calculate importance weights. We denote the importance weight of a trajectory that consists only a single transition as

$$\rho_t \stackrel{\text{def}}{=\!=} W_t^{t+1} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}. \tag{2.34}$$

Therefore, we can rewrite W_k^h in the following way:

$$W_t^h = \prod_{k=t}^{h-1} W_k^{k+1} = \prod_{k=t}^{h-1} \rho_k. \tag{2.35}$$

We denote the importance weight of a trajectory ending in a full termination as $W_t \stackrel{\text{def}}{=\!=} W_t^{T(t)}$.

We require the following assumption for all importance sampling estimators. We call this the assumption of *coverage*.

Assumption 1 (Coverage)**.** *For any state s and action a and two policies π and μ, whenever $\pi(a|s) > 0$, we have $\mu(a|s) > 0$. Equivalently, whenever $\mu(a|s) = 0$, we have $\pi(a|s) = 0$.*

Due to this coverage, it is possible to rescale the samples in such a way as if they were weighted according to the probability distribution induced by the target policy rather than by the behavior policy. Following lemmas exemplify that.

Lemma 5 (Importance weight)**.** *Given Assumption 1, the following holds for $h \geq t$:*

$$\mathrm{E}_\mu \left[W_t^h \middle| S_t = s \right] = 1. \tag{2.36}$$

Proof. Using the definition of expectations, we can write:

$$\mathrm{E}_\mu \left[W_t^h \middle| S_t = s \right] = \sum_{\substack{s_t = s, \\ a_t, \cdots, s_h}} \left(\frac{\prod_{k=t}^{h-1} \pi(a_k|s_k)}{\prod_{k=t}^{h-1} \mu(a_k|s_k)} \right) \prod_{k=t}^{h-1} \mu(a_k|s_k) p(s_{k+1}|s_k, a_k) \tag{2.37}$$

$$= \sum_{\substack{s_t = s, \\ a_t, \cdots, s_h}} \left(\prod_{k=t}^{h-1} \pi(a_k|s_k) \right) \frac{\prod_{k=t}^{h-1} \mu(a_k|s_k) p(s_{k+1}|s_k, a_k)}{\prod_{k=t}^{h-1} \mu(a_k|s_k)} \tag{2.38}$$

$$= \sum_{\substack{s_t = s, \\ a_t, \cdots, s_h}} \prod_{k=t}^{h-1} \pi(a_k|s_k) p(s_{k+1}|s_k, a_k) \tag{2.39}$$

$$= 1. \tag{2.40}$$

Here the last step follows because the sum iterates over the support of μ, and it covers the support of π. □

Corollary 1. *Given Assumption 1, the following holds:*

$$\mathrm{E}_\mu \left[\rho_t \middle| S_t = s \right] = 1. \tag{2.41}$$

Proof. It follows from Lemma 5 with $h = t + 1$, and the definition of ρ_t in (2.34). □

Lemma 6 (Adjustment by importance sampling)**.** *Given Assumption 1, the following holds for a random variable X, which is a function of the trajectory L_t^l, and for constant integers t, h, l such that $h \geq l > t$:*

$$\mathrm{E}_\mu \left[W_t^h X \middle| S_t = s \right] = \mathrm{E}_\pi \left[X \middle| S_t = s \right]. \tag{2.42}$$

Proof. The proof follows similar steps as in the proof of Lemma 5:

$$\mathrm{E}_\mu \left[W_t^h X \middle| S_t = s \right] = \sum_{\substack{s_t = s, x, \\ a_t, \cdots, s_h}} x \left(\frac{\prod_{k=t}^{h-1} \pi(a_k|s_k)}{\prod_{k=t}^{h-1} \mu(a_k|s_k)} \right) \prod_{k=t}^{h-1} \mu(a_k|s_k) p(s_{k+1}|s_k, a_k) \tag{2.43}$$

$$= \sum_{\substack{s_t = s, x, \\ a_t, \cdots, s_h}} x \left(\prod_{k=t}^{h-1} \pi(a_k|s_k) \right) \frac{\prod_{k=t}^{h-1} \mu(a_k|s_k) p(s_{k+1}|s_k, a_k)}{\prod_{k=t}^{h-1} \mu(a_k|s_k)} \tag{2.44}$$

$$= \sum_{\substack{s_t=s,x,\\ a_t,\cdots,s_h}} x \prod_{k=t}^{h-1} \pi(a_k|s_k) p(s_{k+1}|s_k, a_k) \tag{2.45}$$

$$= \mathrm{E}_\pi \left[X \middle| S_t = s \right]. \tag{2.46}$$

□

The last lemma shows that a sample X drawn according to μ can be seen as to be drawn according to π after scaling with importance weight W_t^h in the sense that this scaled sample is an unbiased estimate under the desired distribution π.

2.9 Conclusions

In this chapter, we described the general value function framework for specifying long-term policy-contingent predictions and explained it with an illustrative example. GFVs are one of the most powerful specifications of long-term policy-contingent prediction tasks. Most the off-policy algorithms we develop in this work use this framework for problem formulation. In some cases, we have adopted the simpler continuing task setting, which is a special case under this framework. Moreover, we discuss and provide the background for various categories of task settings and computational goals of learning algorithms that we will later use in this thesis.

Chapter 3

Algorithmic Equivalences for Incremental Reinforcement Learning [1]

In this chapter, we establish the notion of algorithmic equivalences for learning algorithms. Our goal is to understand how to derive equivalent and efficient reinforcement learning algorithms from expensive ones and investigate some of the cases where such derivations are problematic. Derivation of a computationally efficient algorithm with the same outcomes as an expensive algorithm is a recurring theme in this thesis. There are precedences of such equivalences in reinforcement learning prior to this thesis, but typically such equivalences were realized in hindsight, but not through derivations. In some other cases, such equivalences were fortuitous, lacking a general understanding of such phenomena. In this chapter, we discuss the elements fundamental to algorithmic equivalences in the context of reinforcement learning so that computationally efficient algorithms can be obtained systematically through derivations. We build up our understanding from a series of examples in different reinforcement learning settings. The novel techniques for deriving algorithms through algorithmic equivalence are provided in later chapters, whereas this chapter sets the stage for them.

3.1 Algorithmic Equivalences

Algorithmic equivalences are about having different algorithms with different computational requirements producing the same outcomes. Such equivalences are quite common in classical problems of computing science such as string search, sorting, and shortest path problems. For example, there are several algorithms for sorting a list of items such as bubble sort, merge sort, and quick sort, each of which produces the same outcome. However, the worst

[1] The core concepts presented in this chapter are simplifications and illustrations of the ideas developed in published papers coauthored by this author (Mahmood, van Hasselt & Sutton 2014, Mahmood & Sutton 2015).

case computational complexity in terms of the number of items is $O(n^2)$ for bubble and quick sort, whereas it is $O(n \log n)$ for merge sort (Cormen 2009).

Algorithmic equivalences play an important role in machine learning problems. Often a machine learning problem is formulated in terms of optimizing an objective. This objective is specified either in an asymptotic sense as in mean squared error minimization (Hastie et al. 2001) or in terms of a finite number of samples as in empirical risk minimization (Vapnik 1998). Algorithmic equivalences can be achieved in both forms of machine learning problems.

In machine learning problems with asymptotic objectives, it is possible to have multiple algorithms with different computational requirements to achieve asymptotic equivalence. For example, in mean squared error minimization of supervised learning problems, both the Least-Squares (LS) algorithm and Stochastic Gradient Descent (SGD) can be shown to converge to the minimum of the mean squared error under some natural conditions (Eicker 1963, Clarkson 1993). Such an equivalence between LS and SGD is often utilized to argue in favor of using SGD as it is computationally cheaper than LS. However, the equivalence is only asymptotic, and their rates of convergence are different (Bousquet & Bottou 2008), providing a trade-off between computational and sample efficiency.

There are many examples of algorithmic equivalences in the finite-sample empirical risk minimization problems. Many works in the regret minimization literature focus on incorporating existing algorithms into generalized frameworks. By doing so, analyses available for a particular framework become readily applicable to the incorporated algorithms. For example, SGD has been shown to be an instantiation of Online Mirror Descent (Srebro, Sridharan & Tewari 2011). Hazan and Kale (2008) showed that the Follow the Regularized Leader (FTRL) and the Follow the Lazy Projected Leader produce identical outcomes. McMahan (2011) showed that the proximal FTRL and the composite-objective Mirror Descent with a certain form of loss and regularizer function produce equivalent outcomes on a per-sample basis.

In reinforcement learning problems, algorithmic equivalences appear asymptotically as well as in a finite-sample form. Many reinforcement learning algorithms can be shown to converge to the same asymptotic solution. For example, the original TD(λ) and the gradient-based TD algorithms converge to the minimum of the mean squared projected Bellman error (Sutton et al. 2009, Maei 2011). The most profound influence of algorithmic equivalences in reinforcement learning is in the per-sample form. Due to such equivalences, it is possible to apply some of the well-known supervised learning algorithms to reinforcement learning problems in a computationally efficient manner. For example, two classical reinforcement learning algorithms: TD(λ) with accumulating traces and TD(λ) with replacing traces produce the same outcome as two different forms of Monte Carlo algorithms (Sutton 1988, Singh & Sutton 1996). Later LSTD(λ) was also shown to achieve equivalence with the supervised LS algorithm (Boyan 2002).

In this thesis, we consider algorithmic equivalences that occur on a per-sample basis. We are specifically interested in developing algorithmic ideas in supervised learning problems and applying them to reinforcement learning problems in a computationally efficient manner. We complete this section by providing a simple example of algorithmic equivalences.

Example of algorithmic equivalence: Sample average

Consider a process generating a series of samples $\{X_k\}_{k=1}^{\infty}, X_k \in \mathbb{R}$, one at a time. We would like to compute the sample average of these numbers after each sample is presented. The sample average for n samples is defined as

$$V_n \stackrel{\text{def}}{=\!=} \frac{X_1 + X_2 + \cdots + X_n}{n}. \tag{3.1}$$

We could use (3.1) directly to compute sample averages every time a new sample is presented. However, it will require retaining all the samples seen so far, and the cost for computing sample averages will increase as more samples are seen. It is possible to compute sample average more efficiently. One way would be to compute and store incrementally the sum of the samples and the number of samples seen so far:

$$S_n \stackrel{\text{def}}{=\!=} S_{n-1} + X_n; \qquad C_n \stackrel{\text{def}}{=\!=} C_{n-1} + 1. \tag{3.2}$$

Then the sample average after each sample can be computed by dividing the sum by the number of samples: $V_n = \frac{S_n}{C_n}$. After a new sample arrives, both the summation and the number of samples would be updated to recompute the sample average.

Yet another way is to compute the sample average directly using the previous average:

$$V_n = \frac{S_n}{C_n} = \frac{S_{n-1} + X_n}{C_n} = \frac{C_n - 1}{C_n}\frac{S_{n-1}}{C_{n-1}} + \frac{X_n}{C_n} = \left(1 - \frac{1}{C_n}\right) V_{n-1} + \frac{X_n}{C_n} \tag{3.3}$$

$$= V_{n-1} + \frac{1}{C_n}\left(X_n - V_{n-1}\right). \tag{3.4}$$

These three different ways of computing sample averages lead to three different algorithms with different computational and memory requirements. For n samples, the first algorithm requires $O(n^2)$ total computation and $O(n)$ memory. Therefore, each time a sample arrives, more computation and memory are being required to recompute the average. The second and the third algorithms require total $O(n)$ computation and do not require storing past samples, needing only $O(1)$ memory at all time. The exact amount of memory and computation is slightly different between the second and the third algorithms.

This example constitutes one of the simplest occurrences of algorithmic equivalences for learning algorithms. It is also an example of deriving computationally efficient algorithms from an expensive one. In this case, the derivation of the second algorithm is trivial, and the derivation of the third algorithm is also straightforward. However, for more complicated algorithms, derivation of equivalent efficient algorithms can become difficult, or even impossible.

3.2 Equivalences for the Method of Least-Squares

In this section, we explore algorithms that compute the least-squares solution in different reinforcement learning settings and derive strictly-incremental algorithms with equivalent outcomes in different online learning settings.

The method of least-squares is best motivated in an off-line learning setting. Given a set of n samples $\{(G_{t_k}, \boldsymbol{\phi}_{t_k})\}_{k=1}^{n}$, the goal is to compute the parameter vector $\boldsymbol{\theta}$ that minimizes the total squared error:

$$\boldsymbol{\theta}_n = \arg\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{k=1}^{n} \left(G_{t_k} - \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t_k} \right)^2 . \tag{3.5}$$

Let us define $\boldsymbol{\Phi}_n \in \mathbb{R}^{n \times m}$ to be a matrix, where the ith row is $\boldsymbol{\phi}_{t_i}^\top$, and $\mathbf{g}_n \in \mathbb{R}^n$ a column vector containing the returns. Hence,

$$\boldsymbol{\Phi}_n^\top = [\boldsymbol{\phi}_{t_1}, \cdots, \boldsymbol{\phi}_{t_n}] \tag{3.6}$$

$$\mathbf{g}_n^\top = [G_{t_1}, \cdots, G_{t_n}] . \tag{3.7}$$

Then the solution can be computed in the following way:

$$\boldsymbol{\theta}_n = \left(\boldsymbol{\Phi}_n^\top \boldsymbol{\Phi}_n \right)^{-1} \boldsymbol{\Phi}_n^\top \mathbf{g}_n, \tag{3.8}$$

assuming $\boldsymbol{\Phi}_n^\top \boldsymbol{\Phi}_n$ is invertible. This matrix may not always be invertible. This problem can be avoided by computing the solution in the following way:

$$\boldsymbol{\theta}_n = \left(\boldsymbol{\Phi}_n^\top \boldsymbol{\Phi}_n + \epsilon \mathbf{I} \right)^{-1} \boldsymbol{\Phi}_n^\top \mathbf{g}_n, \tag{3.9}$$

where $\mathbf{I} \in \mathbb{R}^{m \times m}$ is the identity matrix, and $\epsilon > 0$ is a regularization parameter. We call this the *Least-Squares* (LS) algorithm.

Let us consider using the LS algorithm in the per-trajectory learning setting. In this case, the learner is expected to produce a series of learned estimates: $\{\boldsymbol{\theta}_k\}_{k=1}^{\infty}$, where the kth estimate is produced after the kth sample $(G_{t_k}, \boldsymbol{\phi}_{t_k})$ is received. A naive way of using LS in this setting would be to use (3.9) after each sample is presented. When n samples are presented, using (3.9) would require $O(nm^2)$ to compute $\boldsymbol{\Phi}_n^\top \boldsymbol{\Phi}_n + \epsilon \mathbf{I}$, $O(m^3)$ to compute the matrix inversion, $O(nm)$ to compute $\boldsymbol{\Phi}_n^\top \mathbf{g}_n$, and $O(m^2)$ to compute the final matrix-vector multiplication. Hence, the total per-sample computation is $O(nm^2)$ considering $n \gg m$. Using LS this way also requires retaining all the samples seen so far. Therefore, the input memory complexity of LS is $O(nm^2)$. Both the memory and per-sample computation increases with the number of samples. The LS algorithm is not strictly incremental when applied to the per-trajectory learning setting.

However, through a small modification we can derive an algorithm that can be computed strictly incrementally in the per-trajectory learning setting. Note that both the matrix

Algorithm Per-Trajectory-Incremental-LS($\mathbf{A}$, $\mathbf{b}$, G, $\boldsymbol{\phi}$)

$\mathbf{A} \leftarrow \mathbf{A} + \boldsymbol{\phi}\boldsymbol{\phi}^\top$
$\mathbf{b} \leftarrow \mathbf{b} + G\boldsymbol{\phi}$
$\boldsymbol{\theta} \leftarrow \mathbf{A}^{-1}\mathbf{b}$
return $(\boldsymbol{\theta}, \mathbf{A}, \mathbf{b})$

Table 3.1: The Per-Trajectory Incremental LS algorithm: It computes the least-squares solution strictly incrementally on a per-trajectory basis.

$\boldsymbol{\Phi}_n^\top \boldsymbol{\Phi}_n$ and the vector $\boldsymbol{\Phi}_n^\top \mathbf{g}_n$ can be computed strictly incrementally in the following way:

$$\mathbf{A}_n = \boldsymbol{\Phi}_n^\top \boldsymbol{\Phi}_n + \epsilon \mathbf{I} = \sum_{k=1}^{n} \boldsymbol{\phi}_{t_k} \boldsymbol{\phi}_{t_k}^\top + \epsilon \mathbf{I} = \mathbf{A}_{n-1} + \boldsymbol{\phi}_{t_n} \boldsymbol{\phi}_{t_n}^\top; \qquad \mathbf{A}_0 = \epsilon \mathbf{I}, \tag{3.10}$$

$$\mathbf{b}_n = \boldsymbol{\Phi}_n^\top \mathbf{g}_n = \sum_{k=1}^{n} G_{t_k} \boldsymbol{\phi}_{t_k} = \mathbf{b}_{n-1} + G_{t_n} \boldsymbol{\phi}_{t_n}; \qquad \mathbf{b}_0 = \mathbf{0}. \tag{3.11}$$

Therefore, after each sample $(G_{t_n}, \boldsymbol{\phi}_{t_n})$ is received, $\mathbf{A}_n \in \mathbb{R}^{m \times m}$ and $\mathbf{b}_n \in \mathbb{R}^m$ can be computed based on this sample, $\mathbf{A}_{n-1}$, and $\mathbf{b}_{n-1}$. Then the estimate can be computed as

$$\boldsymbol{\theta}_n = \mathbf{A}_n^{-1} \mathbf{b}_n. \tag{3.12}$$

This way we only need to store $\mathbf{A}_n \in \mathbb{R}^{m \times m}$ and $\mathbf{b}_n \in \mathbb{R}^m$, and its per-sample computation is $O(m^3)$, neither of which depend on the number of samples. Therefore, this modified algorithm is strictly incremental in the per-trajectory learning setting. We call this algorithm *Per-Trajectory Incremental LS* and summarize it in Table 3.1. We can generate the sequence of estimates $\{\boldsymbol{\theta}_k\}_{k=1}^{\infty}$ by invoking Per-Trajectory Incremental LS after each new sample is received:

$$\begin{aligned}
(\boldsymbol{\theta}_1, \mathbf{A}_1, \mathbf{b}_1) &\leftarrow \text{Per-Trajectory-Incremental-LS}(\mathbf{A}_0, \mathbf{b}_0, G_{t_1}, \boldsymbol{\phi}_{t_1}), \\
(\boldsymbol{\theta}_2, \mathbf{A}_2, \mathbf{b}_2) &\leftarrow \text{Per-Trajectory-Incremental-LS}(\mathbf{A}_1, \mathbf{b}_1, G_{t_2}, \boldsymbol{\phi}_{t_2}), \\
&\vdots \\
(\boldsymbol{\theta}_n, \mathbf{A}_n, \mathbf{b}_n) &\leftarrow \text{Per-Trajectory-Incremental-LS}(\mathbf{A}_{n-1}, \mathbf{b}_{n-1}, G_{t_n}, \boldsymbol{\phi}_{t_n}).
\end{aligned}$$

From (3.10) and (3.11) it is clear that the Per-Trajectory Incremental LS produces the same outcomes as the batch LS algorithm described by (3.9).

Now, let us consider using the Per-Trajectory Incremental LS in the real-time learning setting. After t transitions, the available data in this setting is $\boldsymbol{\phi}_1, A_1, R_2, \boldsymbol{\phi}_2, \cdots, A_t, R_{t+1}, \boldsymbol{\phi}_{t+1}$. On the other hand, Per-Trajectory Incremental LS expects a sequence of trajectories as inputs. How can we use the available data so that the LS algorithm can be applied? One way would be to produce a sequence of overlapping trajectories from the single trajectory available in the real-time setting. Then from each of these trajectories, a return can be

computed. An additional complication is that, the trajectory might not be complete at a particular time step $t+1$. This can be resolved by computing *interim* returns in the following way:

$$
\begin{aligned}
G_1^{t+1} &= R_2 + \gamma_2 R_3 + \gamma_2\gamma_3 R_4 \cdots + \prod_{k=2}^{t} \gamma_k R_{t+1} + \prod_{k=2}^{t+1} \gamma_k P_{t+1}, \\
G_2^{t+1} &= \gamma_2 R_3 + \gamma_2\gamma_3 R_4 \cdots + \prod_{k=2}^{t} \gamma_k R_{t+1} + \prod_{k=2}^{t+1} \gamma_k P_{t+1}, \\
&\vdots \\
G_t^{t+1} &= R_{t+1} + \gamma_{t+1} P_{t+1},
\end{aligned}
$$

where G_k^{t+1} is the corrected truncated return from state S_k computed using data up to step $t+1$. A guess P_{t+1} is used as a proxy for the data not seen so far. This guess can be based on prior estimates. Then these interim returns can be passed to Per-Trajectory Incremental LS in sequence to produce the estimate $\boldsymbol{\theta}_{t+1}$. If a termination occurs at time step $t+1$, that is $\gamma_{t+1} = 0$, then P_{t+1} is fully discounted and the effect of the guess on the return goes away.

Interim returns defined in the above form are often called *n-step backups*. Interim returns can be much more complex, for example, in the form of an exponentially weighted average of different multistep backups. In that case, the effect of the guess may not completely go away even upon full termination. Such targets are called *bootstrapping* targets, as they are formed by bootstrapping on value estimates for other states. We avoid such intricacies here by considering only simple n-step backups and assuming that the guesses are independent of the estimates.

The process of computing a learned prediction after $t+1$ time step using the Per-Trajectory Incremental LS algorithm is given in Table 3.2. We call this *Real-Time LS*, as it produces learned estimates on a real-time basis. However, this algorithm is not strictly incremental. After t transitions, it requires $O(t)$ input memory and $O(t)$ work memory for storing t interim returns. At each time t when a sample arrives, the computational complexity of Real-Time LS is $O(t)$ for computing the interim returns and $O(t)$ for invoking the Per-Trajectory Incremental LS algorithm t times. Clearly, the memory and per-sample computation of this algorithm increase as samples increase. Although Per-Trajectory Incremental LS is strictly incremental in the per-trajectory learning setting, directly invoking it does not produce a strictly incremental algorithm in the real-time learning setting.

Due to the overlapping and incremental nature of interim returns, it is actually possible to derive a strictly incremental algorithm that produces the same outcomes as the Per-Trajectory Incremental LS. The key is to write $\mathbf{A}_t$ and $\mathbf{b}_t$ recursively using only the newest sample. The matrix $\mathbf{A}_t$ can already be computed strictly incrementally on a real-time basis using (3.10). In order to compute $\mathbf{b}_t$ strictly incrementally, let us first consider a simplified

Algorithm Real-Time-LS$((\gamma_1, \boldsymbol{\phi}_1), \{(R_k, \gamma_k, \boldsymbol{\phi}_k)\}_{k=2}^{t+1}, P_{t+1})$

$G_1 \leftarrow G_2 \leftarrow \cdots \leftarrow G_t \leftarrow 0$
$G_t \leftarrow P_{t+1}$
for k from t downto 1 **do**
  $G_k \leftarrow R_{k+1} + \gamma_{k+1} G_{k+1}$
end for
$\mathbf{A} \leftarrow \epsilon \mathbf{I}$, $\mathbf{b} \leftarrow \mathbf{0}$
for $k = 1$ to t **do**
  $(\boldsymbol{\theta}, \mathbf{A}, \mathbf{b}) \leftarrow$ Per-Trajectory-Incr-LS$(\mathbf{A}, \mathbf{b}, G_k, \boldsymbol{\phi}_k)$
end for
return $\boldsymbol{\theta}$

Table 3.2: The Real-Time LS algorithm: It computes the least-squares solution on a real-time basis but not strictly incrementally.

case where there is no discounting and guesses: $\gamma_i = 1, P_i = 0, \forall i > 0$. Then we can write $\mathbf{b}_n$ as follows:

$$\mathbf{b}_t = \sum_{k=1}^{t} G_k^{t+1} \boldsymbol{\phi}_k = \sum_{k=1}^{t} \left(\sum_{i=k}^{t} R_{i+1} \right) \boldsymbol{\phi}_k. \tag{3.13}$$

In the real-time learning setting, we seek incrementality in terms of transitions. There are two summations in the above expression, where the outer summation iterates over different interim returns and the inner summation iterates over different transitions within a return. When a new transition occurs, the above expression will use the new sample $O(t)$ times to compute $\mathbf{b}_{t+1}$, as this new sample needs to contribute to each interim return. In order to reduce the computation involving the new sample to $O(1)$, we need the outer sum to iterate through different transitions.

The following identity enables us to switch the order of the two summations, so that the outer summation iterates over transitions, bringing us closer to a real-time incrementality (Graham et al. 1989, page 36).

Lemma 7 (Double-summation identity)**.** *For $t \geq 1$, the following holds:*

$$\sum_{a=1}^{t} \sum_{b=a}^{t} f(a, b) = \sum_{a=1}^{t} \sum_{b=1}^{a} f(b, a),$$

where the function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^{c \times d}$, with $c, d \geq 1$, takes two natural numbers as inputs and produces a matrix as an output.

Note that, by allowing $c, d \geq 1$, the function f can produce a vector or a scalar as well.

By applying Lemma 7 to (3.13), we then get:

$$\mathbf{b}_t = \sum_{k=1}^{t} \left(\sum_{i=k}^{t} R_{i+1} \right) \boldsymbol{\phi}_k = \sum_{k=1}^{t} \sum_{i=1}^{k} R_{k+1} \boldsymbol{\phi}_i = \sum_{k=1}^{t} R_{k+1} \sum_{i=1}^{k} \boldsymbol{\phi}_i. \tag{3.14}$$

This way we now have the new sample involved only in the outer summation. The inner summation does not have to be recomputed after each transition because the new sample is involved in it only $O(1)$ times. The inner summation can be computed incrementally as:

$$\mathbf{e}_k = \sum_{i=1}^{k} \boldsymbol{\phi}_i = \sum_{i=1}^{k-1} \boldsymbol{\phi}_i + \boldsymbol{\phi}_k = \mathbf{e}_{k-1} + \boldsymbol{\phi}_k; \qquad \mathbf{e}_0 = \mathbf{0}.$$

Then the vector $\mathbf{b}_t$ can be updated incrementally as:

$$\mathbf{b}_t = \sum_{k=1}^{t} R_{k+1} \mathbf{e}_k = \sum_{k=1}^{t-1} R_{k+1} \mathbf{e}_k + R_{t+1} \mathbf{e}_t = \mathbf{b}_{t-1} + R_{t+1} \mathbf{e}_t.$$

For interim returns with arbitrary discounting and guesses, we can apply Lemma 7 in a similar manner and bring the summation over transitions outside:

$$\begin{aligned}
\mathbf{b}_t &= \sum_{k=1}^{t} G_k^{t+1} \boldsymbol{\phi}_k && (3.15)\\
&= \sum_{k=1}^{t} \left(\sum_{i=k}^{t} \prod_{j=k+1}^{i} \gamma_j R_{i+1} + \prod_{j=k+1}^{t+1} \gamma_j P_{t+1} \right) \boldsymbol{\phi}_k && (3.16)\\
&= \sum_{k=1}^{t} \sum_{i=1}^{k} \prod_{j=i+1}^{k} \gamma_j R_{k+1} \boldsymbol{\phi}_i + \sum_{k=1}^{t} \prod_{j=k+1}^{t+1} \gamma_j P_{t+1} \boldsymbol{\phi}_k && (3.17)\\
&= \sum_{k=1}^{t} R_{k+1} \sum_{i=1}^{k} \prod_{j=i+1}^{k} \gamma_j \boldsymbol{\phi}_i + \gamma_{t+1} P_{t+1} \sum_{k=1}^{t} \prod_{j=k+1}^{t} \gamma_j \boldsymbol{\phi}_k && (3.18)\\
&= \sum_{k=1}^{t} R_{k+1} \mathbf{e}_k + \gamma_{t+1} P_{t+1} \mathbf{e}_t, && (3.19)
\end{aligned}$$

where $\mathbf{e}_t$ can be updated incrementally as

$$\mathbf{e}_t = \sum_{k=1}^{t} \prod_{j=k+1}^{t} \gamma_j \boldsymbol{\phi}_k = \gamma_t \sum_{k=1}^{t-1} \prod_{j=k+1}^{t-1} \gamma_j \boldsymbol{\phi}_k + \boldsymbol{\phi}_t = \gamma_t \mathbf{e}_{t-1} + \boldsymbol{\phi}_t; \qquad \mathbf{e}_0 = \mathbf{0}. \tag{3.20}$$

Then a strictly incremental update of $\mathbf{b}_t$ can be derived in the following way:

$$\begin{aligned}
\mathbf{b}_t &= \sum_{k=1}^{t-1} R_{k+1} \mathbf{e}_k + R_{t+1} \mathbf{e}_t + \gamma_t P_t \mathbf{e}_{t-1} - \gamma_t P_t \mathbf{e}_{t-1} + \gamma_{t+1} P_{t+1} \mathbf{e}_t && (3.21)\\
&= \sum_{k=1}^{t-1} R_{k+1} \mathbf{e}_k + \gamma_t P_t \mathbf{e}_{t-1} + R_{t+1} \mathbf{e}_t + \gamma_{t+1} P_{t+1} \mathbf{e}_t - P_t \left(\mathbf{e}_t - \boldsymbol{\phi}_t \right) && (3.22)
\end{aligned}$$

Algorithm Real-Time-Incremental-LS($\boldsymbol{\phi}, R, \gamma, P, \gamma', P', \mathbf{A}, \mathbf{b}, \mathbf{e}$)

$\mathbf{A} \leftarrow \mathbf{A} + \boldsymbol{\phi}\boldsymbol{\phi}^\top$
$\mathbf{e} \leftarrow \gamma\mathbf{e} + \boldsymbol{\phi}$
$\mathbf{b} \leftarrow \mathbf{b} + (R + \gamma' P' - P)\,\mathbf{e} + P\boldsymbol{\phi}$
$\boldsymbol{\theta} \leftarrow \mathbf{A}^{-1}\mathbf{b}$
return $(\boldsymbol{\theta}, \mathbf{A}, \mathbf{b}, \mathbf{e})$

Table 3.3: The Real-Time Incremental LS algorithm: It computes the least-squares solution strictly incrementally on a real-time basis.

$$= \mathbf{b}_{t-1} + (R_{t+1} + \gamma_{t+1} P_{t+1} - P_t)\,\mathbf{e}_t + P_t \boldsymbol{\phi}_t. \tag{3.23}$$

The resulting algorithm is given in Table 3.3. We call this algorithm the Real-Time Incremental LS algorithm. This algorithm requires only $O(1)$ input and work memory and $O(1)$ computation. We can produce a series of learned estimates by invoking the Real-Time Incremental LS algorithm using new samples each time a transition occurs:

$$\begin{aligned}
(\mathbf{A}_0, \mathbf{b}_0, \mathbf{e}_0) &\leftarrow (\epsilon\mathbf{I}, \mathbf{0}, \mathbf{0}),\\
(\boldsymbol{\theta}_2, \mathbf{A}_1, \mathbf{b}_1, \mathbf{e}_1) &\leftarrow \text{Real-Time-Incremental-LS}(\boldsymbol{\phi}_1, R_2, \gamma_1, P_1, \gamma_2, P_2, \mathbf{A}_0, \mathbf{b}_0, \mathbf{e}_0),\\
(\boldsymbol{\theta}_3, \mathbf{A}_2, \mathbf{b}_2, \mathbf{e}_2) &\leftarrow \text{Real-Time-Incremental-LS}(\boldsymbol{\phi}_2, R_3, \gamma_2, P_2, \gamma_3, P_3, \mathbf{A}_1, \mathbf{b}_1, \mathbf{e}_1),\\
&\vdots\\
(\boldsymbol{\theta}_{t+1}, \mathbf{A}_t, \mathbf{b}_t, \mathbf{e}_t) &\leftarrow \text{Real-Time-Incremental-LS}(\boldsymbol{\phi}_t, R_{t+1}, \gamma_t, P_t, \gamma_{t+1}, P_{t+1}, \mathbf{A}_{t-1}, \mathbf{b}_{t-1}, \mathbf{e}_{t-1}).
\end{aligned}$$

From the above derivation, it is evident that the sequence of estimates $\{\boldsymbol{\theta}_k\}_{k=1}^{\infty}$ computed by the Real-Time Incremental LS algorithm are equivalent to those computed by the Real-Time LS algorithm.

Strictly incremental computation of the least-squares solution on a real-time basis is not novel. Boyan (2002) showed that LSTD(1) without discounting computes the least-squares estimate at the end of each episode. Geist and Scherrer (2014) used Lemma 7 to develop several least-squares-based RL algorithms with strictly incremental updates and real-time equivalences. Their algorithms differ by the way bootstrapping estimates are used in place of the guesses. The trace vector $\mathbf{e}_n$ we obtained in our Real-Time Incremental LS algorithm is known as the *accumulating traces* and is present in most forms of least-squares algorithms with strictly incremental updates.

3.3 Equivalences for Stochastic Gradient Descent

We explore how to derive stochastic gradient descent (SGD) incrementally on a real-time basis. LS and SGD are fundamentally different in the way they are updated. LS arises from batch learning settings, whereas SGD is inherently rooted in online learning settings. On the

Algorithm Real-Time-SGD$(\boldsymbol{\theta}_1, (\gamma_1, \boldsymbol{\phi}_1), \{(R_k, \gamma_k, \boldsymbol{\phi}_k, \alpha_{k-1})\}_{k=2}^{t+1}, P_{t+1})$

$G_1 \leftarrow G_2 \leftarrow \cdots \leftarrow G_t \leftarrow 0$
$G_t \leftarrow P_{t+1}$
for k from t downto 1 **do**
  $G_k \leftarrow R_{k+1} + \gamma_{k+1} G_{k+1}$
end for
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_1$
for $k = 1$ to t **do**
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_k \left(G_k - \boldsymbol{\theta}^\top \boldsymbol{\phi}_k\right) \boldsymbol{\phi}_k$
end for
return $\boldsymbol{\theta}$

Table 3.4: The Real-Time SGD algorithm: It computes estimates on a real-time basis, but not strictly incrementally.

other hand, LS involves straightforward summations whereas SGD involves a complex form of exponentially moving average. In the per-trajectory learning setting, the SGD update is as follows:

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \alpha_t \left(G_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t\right) \boldsymbol{\phi}_t, \tag{3.24}$$

where $\alpha_t > 0$ is a scalar step-size parameter chosen at step t. The sequence of step sizes is often chosen to be constant or reduced with time using a deterministic schedule. As with LS, we can apply SGD directly in the real-time learning setting by taking a single trajectory and converting them into overlapping interim returns. The complete process is given in Table 3.4. We call this algorithm *Real-Time SGD*. Just like Real-Time LS (Table 3.2), Real-Time SGD requires increasing memory and per-sample computation as the number of samples increases.

In order to update $\boldsymbol{\theta}_t$ incrementally in real-time, we have to unroll $\boldsymbol{\theta}_t$ completely and find a way to express it in terms of the current sample, $\boldsymbol{\theta}_{t-1}$ and other parameters that can be computed and combined using $O(1)$ computation. We again consider the simpler case of no discounting and guesses. We denote the estimate after $t+1$ time steps as $\boldsymbol{\theta}_{t+1}$ and the kth estimate the Real-Time SGD produces internally with horizon $t+1$ as $\boldsymbol{\theta}_k^{t+1}$. It is clear that $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1}^{t+1}$. Then we can unroll $\boldsymbol{\theta}_{t+1}$ as follows:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1}^{t+1} = \boldsymbol{\theta}_t^{t+1} + \alpha_t \left(G_t^{t+1} - \boldsymbol{\phi}_t^\top \boldsymbol{\theta}_t^{t+1}\right) \boldsymbol{\phi}_t \tag{3.25}$$

$$= \underbrace{\left(\mathbf{I} - \alpha_t \boldsymbol{\phi}_t \boldsymbol{\phi}_t^\top\right)}_{\mathbf{F}_t} \boldsymbol{\theta}_t^{t+1} + \alpha_t G_t^{t+1} \boldsymbol{\phi}_t \tag{3.26}$$

$$= \sum_{k=1}^{n} \underbrace{\left(\prod_{i=k+1}^{t} \mathbf{F}_i \right)}_{\mathbf{F}_{k+1}^{t}} \alpha_k G_k^{t+1} \boldsymbol{\phi}_k + \left(\prod_{i=1}^{t} \mathbf{F}_i \right) \boldsymbol{\theta}_1, \qquad \text{where } \mathbf{F}_k = \mathbf{I} - \alpha_k \boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top, \tag{3.27}$$

$$= \sum_{k=1}^{t} \mathbf{F}_{k+1}^{t} \alpha_k \left(\sum_{i=k}^{t} R_{i+1} \right) \boldsymbol{\phi}_k + \mathbf{F}_1^t \boldsymbol{\theta}_1, \qquad \text{where } \mathbf{F}_k^t = \prod_{i=k}^{t} \mathbf{F}_i. \tag{3.28}$$

The estimate of SGD involves double summations as well. In this case, we also have products of matrices involved, producing a complex form of exponentially moving average. By applying Lemma 7 again, we get:

$$\boldsymbol{\theta}_{t+1} = \sum_{k=1}^{t} \sum_{i=k}^{t} \mathbf{F}_{k+1}^{t} \alpha_k R_{i+1} \boldsymbol{\phi}_k + \mathbf{F}_1^t \boldsymbol{\theta}_1 \tag{3.29}$$

$$= \sum_{k=1}^{t} \sum_{i=1}^{k} \mathbf{F}_{i+1}^{t} \alpha_i R_{k+1} \boldsymbol{\phi}_i + \mathbf{F}_1^t \boldsymbol{\theta}_1 \tag{3.30}$$

$$= \sum_{k=1}^{n} R_{k+1} \mathbf{F}_{k+1}^{n} \underbrace{\sum_{i=1}^{k} \mathbf{F}_{i+1}^{k} \alpha_i \boldsymbol{\phi}_i}_{\mathbf{e}_k} + \mathbf{F}_1^t \boldsymbol{\theta}_1 \tag{3.31}$$

$$= \sum_{k=1}^{t} R_{k+1} \mathbf{F}_{k+1}^{t} \mathbf{e}_k + \mathbf{F}_1^t \boldsymbol{\theta}_1 \tag{3.32}$$

$$= \mathbf{F}_t \sum_{k=1}^{t-1} R_{k+1} \mathbf{F}_{k+1}^{t-1} \mathbf{e}_k + \mathbf{F}_t \mathbf{F}_1^{t-1} \boldsymbol{\theta}_1 + R_{t+1} \mathbf{e}_t \tag{3.33}$$

$$= \mathbf{F}_t \boldsymbol{\theta}_t + R_{t+1} \mathbf{e}_t \tag{3.34}$$

$$= \boldsymbol{\theta}_t + R_{t+1} \mathbf{e}_t - \alpha_t \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t \boldsymbol{\phi}_t. \tag{3.35}$$

The trace vector $\mathbf{e}_t$ can be incrementally updated in the following way:

$$\mathbf{e}_t = \sum_{i=1}^{t} \mathbf{F}_{i+1}^{t} \alpha_i \boldsymbol{\phi}_i = \mathbf{F}_t \sum_{i=1}^{t-1} \mathbf{F}_{i+1}^{t-1} \alpha_i \boldsymbol{\phi}_i + \alpha_t \boldsymbol{\phi}_t = \mathbf{F}_t \mathbf{e}_{t-1} + \alpha_t \boldsymbol{\phi}_t \tag{3.36}$$

$$= \mathbf{e}_{t-1} - \alpha_t \mathbf{e}_{t-1}^\top \boldsymbol{\phi}_t \boldsymbol{\phi}_t + \alpha_t \boldsymbol{\phi}_t. \tag{3.37}$$

Therefore, in the general case of interim returns with arbitrary discounting and guesses, the parameter vector $\boldsymbol{\theta}_{n+1}$ can be incrementally updated as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1}^{t+1} = \boldsymbol{\theta}_t^{t+1} \alpha_t \left(G_t^{t+1} - \boldsymbol{\theta}_t^{t+1\top} \boldsymbol{\phi}_t \right) \boldsymbol{\phi}_t \tag{3.38}$$

$$= \underbrace{\left(\mathbf{I} - \alpha_t \boldsymbol{\phi}_t \boldsymbol{\phi}_t^\top \right)}_{\mathbf{F}_t} \boldsymbol{\theta}_t^{t+1} + \alpha_t G_t^{t+1} \boldsymbol{\phi}_t \tag{3.39}$$

$$= \sum_{k=1}^{t} \left(\prod_{i=k+1}^{t} \mathbf{F}_i \right) \alpha_k G_k^{t+1} \boldsymbol{\phi}_k + \left(\prod_{i=1}^{t} \mathbf{F}_i \right) \boldsymbol{\theta}_1 \tag{3.40}$$

$$= \sum_{k=1}^{t} \mathbf{F}_{k+1}^{t} \alpha_k \left(\sum_{i=k}^{t} \prod_{j=k+1}^{i} \gamma_j R_{i+1} + \prod_{j=k+1}^{t+1} \gamma_j P_{t+1} \right) \boldsymbol{\phi}_k + \mathbf{F}_1^t \boldsymbol{\theta}_1 \tag{3.41}$$

$$= \sum_{k=1}^{t} \sum_{i=1}^{k} \mathbf{F}_{i+1}^{t} \boldsymbol{\phi}_i \alpha_i \prod_{j=i+1}^{k} \gamma_j R_{k+1} + \sum_{k=1}^{t} \mathbf{F}_{k+1}^{t} \boldsymbol{\phi}_k \alpha_k \prod_{j=k+1}^{t+1} \gamma_j P_{t+1} + \mathbf{F}_1^t \boldsymbol{\theta}_1 \tag{3.42}$$

$$= \sum_{k=1}^{t} R_{k+1} \mathbf{F}_{k+1}^{t} \underbrace{\sum_{i=1}^{k} \mathbf{F}_{i+1}^{k} \boldsymbol{\phi}_i \alpha_i \prod_{j=i+1}^{k} \gamma_j}_{\mathbf{e}_k} \tag{3.43}$$

$$+ \gamma_{t+1} P_{t+1} \sum_{k=1}^{t} \mathbf{F}_{k+1}^{t} \boldsymbol{\phi}_k \alpha_k \prod_{j=k+1}^{t} \gamma_j + \mathbf{F}_1^t \boldsymbol{\theta}_1 \tag{3.44}$$

$$= \sum_{k=1}^{t} R_{k+1} \mathbf{F}_{k+1}^{t} \mathbf{e}_k + \gamma_{t+1} P_{t+1} \mathbf{e}_t + \mathbf{F}_1^t \boldsymbol{\theta}_1, \tag{3.45}$$

where the trace vector is update as

$$\mathbf{e}_t = \sum_{i=1}^{t} \mathbf{F}_{i+1}^{t} \boldsymbol{\phi}_i \alpha_i \prod_{j=i+1}^{t} \gamma_j = \gamma_t \mathbf{F}_t \sum_{i=1}^{t-1} \mathbf{F}_{i+1}^{t-1} \boldsymbol{\phi}_i \alpha_i \prod_{j=i+1}^{t-1} \gamma_j + \alpha_t \boldsymbol{\phi}_t \tag{3.46}$$

$$= \gamma_t \mathbf{F}_t \mathbf{e}_{t-1} + \alpha_t \boldsymbol{\phi}_t \tag{3.47}$$

$$= \gamma_t \mathbf{e}_{t-1} - \gamma_t \alpha_t \mathbf{e}_{t-1}^{\top} \boldsymbol{\phi}_t \boldsymbol{\phi}_t + \alpha_t \boldsymbol{\phi}_t; \qquad \mathbf{e}_0 = \mathbf{0}. \tag{3.48}$$

Then we can re-write $\boldsymbol{\theta}_{t+1}$ as:

$$\boldsymbol{\theta}_{t+1} = \mathbf{F}_t \sum_{k=1}^{t-1} R_{k+1} \mathbf{F}_{k+1}^{t-1} \mathbf{e}_k + R_{t+1} \mathbf{e}_t + \mathbf{F}_t \mathbf{F}_1^{t-1} \boldsymbol{\theta}_1 \tag{3.49}$$

$$+ \mathbf{F}_t \gamma_t P_t \mathbf{e}_{t-1} - \mathbf{F}_t \gamma_t P_t \mathbf{e}_{t-1} + \gamma_{t+1} P_{t+1} \mathbf{e}_t \tag{3.50}$$

$$= \mathbf{F}_t \left(\sum_{k=1}^{t-1} R_{k+1} \mathbf{F}_{k+1}^{t-1} \mathbf{e}_k + \gamma_t P_t \mathbf{e}_{t-1} + \mathbf{F}_1^{t-1} \boldsymbol{\theta}_1 \right) \tag{3.51}$$

$$+ R_{t+1} \mathbf{e}_t + \gamma_{t+1} P_{t+1} \mathbf{e}_t - P_t \left(\mathbf{e}_t - \alpha_t \boldsymbol{\phi}_t \right) \tag{3.52}$$

$$= \mathbf{F}_t \boldsymbol{\theta}_t + R_{t+1} \mathbf{e}_t + \gamma_{t+1} P_{t+1} \mathbf{e}_t - P_t \left(\mathbf{e}_t - \alpha_t \boldsymbol{\phi}_t \right) \tag{3.53}$$

$$= \boldsymbol{\theta}_t + \left(R_{t+1} + \gamma_{t+1} P_{t+1} - P_t \right) \mathbf{e}_t + \alpha_t \left(P_t - \boldsymbol{\theta}_t^{\top} \boldsymbol{\phi}_t \right) \boldsymbol{\phi}_t. \tag{3.54}$$

The complete algorithm is given in Table 3.5. We call this algorithm the *Real-Time Incremental SGD* algorithm. The memory and the per-sample computation of this algorithm are $O(1)$. We can produce a series of learned estimates by invoking the Real-Time Incremental SGD algorithm using new samples each time a transition occurs:

$$\begin{aligned}
&\mathbf{e}_0 \leftarrow \mathbf{0}, \\
&(\boldsymbol{\theta}_2, \mathbf{e}_1) \leftarrow \text{Real-Time-Incr-SGD}(\boldsymbol{\theta}_1, \boldsymbol{\phi}_1, R_2, \gamma_1, P_1, \gamma_2, P_2, \mathbf{e}_0, \alpha_1), \\
&(\boldsymbol{\theta}_3, \mathbf{e}_2) \leftarrow \text{Real-Time-Incr-SGD}(\boldsymbol{\theta}_2, \boldsymbol{\phi}_2, R_3, \gamma_2, P_2, \gamma_3, P_3, \mathbf{e}_1, \alpha_2),
\end{aligned}$$

Algorithm Real-Time-Incremental-SGD($\boldsymbol{\theta}, \boldsymbol{\phi}, R, \gamma, P, \gamma', P', \mathbf{e}, \alpha$)

$\mathbf{e} \leftarrow \gamma\mathbf{e} - \gamma\alpha\mathbf{e}^\top\boldsymbol{\phi}\boldsymbol{\phi} + \alpha\boldsymbol{\phi}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + (R + \gamma' P' - P)\,\mathbf{e} + \alpha\left(P - \boldsymbol{\theta}^\top\boldsymbol{\phi}\right)\boldsymbol{\phi}$

return $(\boldsymbol{\theta}, \mathbf{e})$

Table 3.5: The Real-Time Incremental SGD algorithm: It computes learned estimates strictly incrementally on a real-time basis.

$$\vdots$$

$$(\boldsymbol{\theta}_{t+1}, \mathbf{e}_t) \leftarrow \text{Real-Time-Incr-SGD}(\boldsymbol{\theta}_t, \boldsymbol{\phi}_t, R_{t+1}, \gamma_t, P_t, \gamma_{t+1}, P_{t+1}, \mathbf{e}_{t-1}, \alpha_t).$$

It is evident from the above derivation that the estimates produced by the Real-Time Incremental SGD are equivalent to those produced by the Real-Time SGD.

A real-time incremental implementation of SGD was first introduced by van Seijen and Sutton (2014, van Seijen et al. 2016). The derivation in their work uses intuitions specific to the non-incremental algorithm and follows a different technique than the one used here. The technique used here is simpler, more mechanistic, and is quite similar to the derivation of the Real-Time Incremental LS. In both cases, we have double summations, an application of Lemma 7, and linear recursions. Although these two algorithms are quite different, and it is natural to expect that their derivations would follow different techniques as well, our derivations here show that the same mechanistic technique can be used to derive both algorithms.

3.4 Counterexamples to Strict Incrementality

We have seen a series of examples with increasing complexity illustrating algorithmic equivalences through derivations in different reinforcement learning settings. A natural question is whether a derivation of such equivalent incremental algorithms is always possible. For that, we consider some hypothetical examples using algebraic expressions that do not constitute any actual learning algorithm.

We consider algebraic expressions using only three basic mathematical operations: summation, multiplication, and division. Let us consider an online learning setting where a sequence of samples $\{X_k\}_{k=1}^{\infty}$ appear one at a time to the learner. A batch algorithm can be defined as a variadic function f_n producing an estimate $\boldsymbol{\theta}_n$ based on the first n samples: $X_1, \cdots, X_n \mapsto \boldsymbol{\theta}_n$. The question is whether there is an equivalent strictly incremental algorithm for f_n, that is, an algorithm producing $\boldsymbol{\theta}_n$ using only $O(1)$ input memory, work memory and per-sample computation.

For concreteness, consider there is a function g that receives only c most recent samples $\{X_k\}_{k=n-c+1}^{n}$, where c is a constant, and an auxiliary parameter $\boldsymbol{\psi}_{n-1}$. Also consider that the function g produces a new update of the auxiliary parameter $\boldsymbol{\psi}_n$ and the estimate $\boldsymbol{\theta}_n$:

$\left(\{X_k\}_{k=n-c+1}^n, \boldsymbol{\psi}_{n-1}\right) \mapsto (\boldsymbol{\theta}_n, \boldsymbol{\psi}_n)$. Then the function g is a strictly incremental algorithm if its computation, the memory requirement for storing $\boldsymbol{\psi}_n$, and the internal working memory is $O(1)$ in terms of the number of samples.

For the real-time incremental least-squares algorithm, $\{X_k\}_{k=n-c+1}^n$ stands for $(\boldsymbol{\phi}_n, R_{n+1}, P_n, \gamma_n, P_{n+1}, \gamma_{n+1})$ and the auxiliary parameter $\boldsymbol{\psi}_n$ stands for $(\mathbf{A}_n, \mathbf{b}_n, \mathbf{e}_n)$. In the case of real-time incremental SGD, $\{X_k\}_{k=n-c+1}^n$ remains the same, and the auxiliary parameter is $(\boldsymbol{\theta}_n, \mathbf{e}_{n-1}, \alpha_n)$. An incremental algorithm either computes a recursive function or can be computed based on a constant number of other recursive functions.

In order to see an example of a batch algorithm which does not have a real-time incremental counterpart, consider the following variadic function computing $\boldsymbol{\theta}_n$ from the sequence $\{X_k\}_{k=1}^n$:

$$\boldsymbol{\theta}_n = (X_n)\,(X_n + X_{n-1})\,(X_n + X_{n-1} + X_{n-2}) \cdots (X_n + \cdots + X_1) = \prod_{k=1}^{n} \sum_{i=k}^{n} X_i. \tag{3.55}$$

Can we compute $\boldsymbol{\theta}_n$ recursively? In order to explore that, consider how to express $\boldsymbol{\theta}_{n+1}$ in terms of $\boldsymbol{\theta}_n$:

$$\boldsymbol{\theta}_{n+1} = \prod_{k=1}^{n+1} \sum_{i=k}^{n+1} X_i = \left(\sum_{i=1}^{n+1} X_i\right) \prod_{k=1}^{n} \left(\sum_{i=k}^{n} X_i + X_n\right) \tag{3.56}$$

$$= \left(\sum_{i=1}^{n+1} X_i\right) \times \sum_{\substack{Z_1, \cdots, Z_n \\ Z_j \in \left\{\sum_{i=j}^n X_i, X_n\right\}}} \prod_{k=1}^{n} Z_k. \tag{3.57}$$

In the above, the first summation can be computed strictly incrementally. The second summation goes through all possible products $\prod_{k=1}^n Z_k$, where the factors are $Z_k \in \{\sum_{i=k}^n X_i, X_n\}$. Only one of the summands in the second summation is $\boldsymbol{\theta}_n$, and there are $2^n - 1$ other summands! We need $O(2^n)$ computation in order to compute $\boldsymbol{\theta}_n$ recursively. A direct computation of $\boldsymbol{\theta}_n$ requires computing each factor $\sum_{i=k}^n X_i$ first, which is $O(n)$ and then forming the product, which is also $O(n)$. To compute $\boldsymbol{\theta}_n$ strictly incrementally, it is not necessary to express $\boldsymbol{\theta}_n$ recursively, but in that case, it must be expressed in terms of $O(1)$ number of auxiliary parameters with recursive updates. It appears that a strictly incremental computation of $\boldsymbol{\theta}_n$ is not possible.

This makes us wonder whether a strictly incremental algorithm is not possible whenever there is a product of sums. For that, consider the following definition of $\boldsymbol{\theta}_n$:

$$\boldsymbol{\theta}_n = (X_1)\,(X_1 + X_2) \cdots (X_1 + X_2 + \cdots + X_n) = \prod_{k=1}^{n} \sum_{i=1}^{k} X_i. \tag{3.58}$$

If we are to compute $\boldsymbol{\theta}_n$ at each step n from the scratch, it requires $O(n)$ computation. However, we can compute $\boldsymbol{\theta}_n$ recursively in the following way:

$$\boldsymbol{\theta}_{n+1} = \prod_{k=1}^{n+1} \sum_{i=1}^{k} X_i = \left(\sum_{i=1}^{n+1} X_i \right) \prod_{k=1}^{n} \sum_{i=1}^{k} X_i = \boldsymbol{\chi}_{n+1} \boldsymbol{\theta}_n, \tag{3.59}$$

where $\boldsymbol{\chi}_n$ is computed incrementally as $\boldsymbol{\chi}_{n+1} = \boldsymbol{\chi}_n + X_{n+1}$. We can compute $\boldsymbol{\theta}_n$ strictly incrementally in this case.

Therefore, it appears that not all products of sums cause difficulty for strictly incremental computation. However, when a new sample affects all the factors in the product of sums, then the difficulty arises.

The following definition of $\boldsymbol{\theta}_n$ constitutes another example where the derivation of a strictly incremental algorithm does not seem possible:

$$\boldsymbol{\theta}_n = \frac{X_1}{X_1 + \cdots + X_n} + \frac{X_2}{X_2 + \cdots + X_n} + \cdots + \frac{X_{n-2}}{X_{n-2} + X_{n-1} + X_n} + \frac{X_{n-1}}{X_{n-1} + X_n} \tag{3.60}$$

$$= \sum_{k=1}^{n-1} \left(\frac{X_k}{\sum_{i=k}^{n} X_i} \right). \tag{3.61}$$

3.5 Equivalence Techniques and Intuitions

After looking at a number of successful and failed derivations, we wonder whether a general technique can be developed that can be used for a large class of algorithms to derive equivalent strictly incremental algorithms. In principle, an algorithm can take an arbitrary form and is not restricted to be an algebraic expression. However, for a restricted class of algorithms, we may have some characterizations whether and how such derivations might be possible.

What does make strict incrementality impossible for algebraic expressions? The following theorem provides us a sufficient condition for non strict incrementality.

Theorem 1 (Condition for non-strict incrementality)**.** *Let $\boldsymbol{\theta}_n$ be an estimate based on n samples described in terms of an algebraic expression. If there exists no expression of $\boldsymbol{\theta}_n$ where the nth sample appears only in $O(1)$ number of places, then this estimate cannot be computed strictly incrementally.*

Proof. A proof by contradiciton is straightforward. Let us consider that $\boldsymbol{\theta}_n$ is an estimate where in each expression the new sample appears in $\omega(1)$ number of places, and $\boldsymbol{\theta}_n$ has a strictly incremental implementation. Let us say the expression corresponding to the strictly incremental implementation consists of $O(h(n))$ places where the nth sample appears. Each of these appearances of the nth sample is associated with a mathematical operator. Therefore, it requires at least $O(h(n))$ computation to produce $\boldsymbol{\theta}_n$. Then $h(n)$ must be a constant function of n, which makes nth sample appear in $O(1)$ places, a contradiction. □

In each of the expressions of $\boldsymbol{\theta}_n$ in (3.55), (3.57) and (3.60), the new sample X_n appears at least in $O(n)$ number of places. It does not readily follow that there are no alternative expressions where $\boldsymbol{\theta}_n$ appears only in $O(1)$ places. However, the following observations would help us further understand the difficulty of deriving strictly incremental implementations of the above equations.

When an expression contains the new sample in $O(n)$ places, it can be transformed into another expression with $O(1)$ appearance of the new sample in certain cases through the distributive law of operators. For example, in the following:

$$X_1X_n + X_2X_n + \cdots + X_{n-1}X_n = X_n\left(X_1 + X_2 + \cdots + X_{n-1}\right),$$

where the number of places the nth sample X_n appears is reduced from n to 1. Here the distributivity of multiplication over addition is used.

In Equation (3.55) and (3.60), the nth sample associates with additions. In the first case, the distributivity of addition over multiplication and in the second case, the left-distributivity of division over addition would have helped. However, none of these distributivity laws hold. This further provides intuitions why strictly incremental implementation in these two examples could not be obtained.

In summary, to have strictly incremental update of $\boldsymbol{\theta}_n$, we have to express the definition of $\boldsymbol{\theta}_n$ in terms of a constant number of appearance of the new sample and a constant number of parameters which themselves have strictly incremental updates. Therefore, each time we have a definition of an estimate $\boldsymbol{\theta}_n$ for which we seek to derive a strictly incremental implementation, we would attempt to reduce the number of times the new sample appears to a constant and express the rest of the expression in terms of other parameters with strictly incremental updates.

As we shall see, such strict incrementally is typically achieved for reinforcement learning algorithms by the distributivity of multiplication over addition, which is the key step behind the switching of the order of the double summations achieved by Lemma 7. To expand its application, this lemma can be further generalized by extending the limits of its sums:

Lemma 8 (Generalized double-summation identity)**.** *For $c \leq t$, $d \leq e$, the following holds:*

$$\sum_{a=c}^{t}\sum_{b=a+d}^{t+e} f(a,b) = \sum_{b=c+d}^{t+e}\sum_{a=c}^{b-d} f(a,b),$$

or alternatively,

$$\sum_{a=c}^{t}\sum_{b=a+d}^{t+e} f(a,b) = \sum_{a=c+d}^{t+e}\sum_{b=c}^{a-d} f(b,a),$$

where the function f takes two natural numbers as inputs and produces a matrix as an output $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^{c\times d}$, with $c, d \geq 1$.

Proof. This theorem can be easily proven by working on the limits of the two sums:

$$[c \leq a \leq t][a + d \leq b \leq t + e] = [c \leq a \leq t][a \leq b - d \leq t + e - d] \tag{3.62}$$

$$= [c \leq a \leq b - d \leq t + e - d] \tag{3.63}$$

$$= [c \leq b - d \leq t + e - d][c \leq a \leq b - d] \tag{3.64}$$

$$= [c + d \leq b \leq t + e][c \leq a \leq b - d], \tag{3.65}$$

which proves the first form.

We can obtain the second form simply by exchanging the variables. □

3.6 Conclusions

In this chapter, we discussed the core concepts involving algorithmic equivalences in reinforcement learning. Our ultimate goal throughout the thesis is to develop algorithms that can be updated strictly incrementally on a real-time basis. We have illustrated how to derive such algorithms using two notable machine learning algorithms: the method of least-squares and stochastic gradient descent. We also described two examples where a strictly incremental update could not be derived. To understand the difficulty in deriving strictly incremental updates, we provided a sufficient condition where a strictly incremental update cannot be derived for a restricted class of algorithms involving algebraic expressions. We concluded by summarizing some intuitions behind the derivations involving algorithmic equivalence in reinforcement learning.

Chapter 4

Tabular Off-policy Algorithms for Value Function Estimation[1]

In this chapter, we introduce two new model-free off-policy algorithms for tabular value function estimation: OIS2 and WIS2, and compare them with existing tabular algorithms. These new tabular algorithms constitute the first of a series of contributions toward overcoming the issue of high variance based on weighted importance sampling. En route, we explore several model-free off-policy algorithms for tabular value function estimation. The tabular representation is the simplest to understand and analyze. Algorithms for this setting form the basis for developing more sophisticated algorithms such as those with function approximation and bootstrapping. Model-free off-policy algorithms all use importance sampling as a core component, and they mainly differ from each other based on how importance sampling is applied to samples. We systematically investigate the existing tabular off-policy algorithms and also develop and study the new ones. The algorithms producing superior performance lay the foundation for off-policy algorithms with linear function approximation developed in subsequent chapters. For simplicity, we assume that each trajectory always terminates fully under the target policy.

4.1 Off-policy Tabular Estimators with Importance Sampling

A tabular estimate of the value for a particular state can be easily computed by generating many returns from that state and averaging them. Such a simple average is directly applicable in the on-policy case. The discrepancy between the target and the behavior policy in off-policy learning makes value estimation more complicated. To account for this discrepancy, off-policy estimators use importance sampling, which is introduced in Section 2.8. There are two most commonly used forms of importance sampling estimators: the average estimator (2.3) and the ratio estimator (2.4). The off-policy estimators we introduce also

[1]The novel algorithms developed in this chapter are based on simplifications of the algorithms developed in a published paper coauthored by this author (Mahmood, van Hasselt & Sutton 2014). The analysis of these algorithms are originally produced for this document.

take one of these forms.

Consider n sample returns $\{G_{t_k}\}_{k=1}^n$ generated from state s following policy μ, and importance weights $\{W_{t_k}\}_{k=1}^n$ of their corresponding trajectories, where $W_t \stackrel{\text{def}}{=\!=} \rho_t^{T(t)}$ defined in Section 2.8. Here, t_k denotes the initial time step of the kth trajectory. For our first pair of estimators, we define $X_k \stackrel{\text{def}}{=\!=} W_{t_k} G_{t_k}$. Then we can easily show that X_k is an unbiased estimate of $v_\pi(s)$, which we formally state below.

Lemma 9 (Unbiasedness of scaled return)**.** *Let Assumption 1 hold. If G_t is a return generated from state s at a given time step t following policy μ and W_t the corresponding importance weight, then $W_t G_t$ is an unbiased estimator of $v_\pi(s)$, that is:*

$$\mathrm{E}_\mu \left[W_t G_t | S_t = s \right] = v_\pi(s). \tag{4.1}$$

Proof. By expanding return G_t, we can write:

$$\mathrm{E}_\mu \left[W_t G_t | S_t = s \right] = \mathrm{E}_\pi \left[G_t | S_t = s \right] = v_\pi(s). \tag{4.2}$$

The first equality follows from Lemma 6. □

The most ordinary form of importance sampling estimator for off-policy estimation is an average of the random variables $W_{t_k} G_{t_k}$. We call this the *ordinary importance sampling estimator* (OIS). Given a sequence of n returns $\{G_{t_k}\}_{k=1}^n$ originated from state s and corresponding importance weights $\{W_{t_k}\}_{k=1}^n$, the ordinary importance sampling estimator $V_n^{\text{OIS}}(s)$ is defined as

$$V_n^{\text{OIS}}(s) \stackrel{\text{def}}{=\!=} \frac{\sum_{k=1}^n W_{t_k} G_{t_k}}{n}. \tag{4.3}$$

A ratio estimator can be formed using (2.4) and random variables $X_k \stackrel{\text{def}}{=\!=} W_{t_k} G_{t_k}$ and $Y_k \stackrel{\text{def}}{=\!=} W_{t_k}$. This estimator is known as the *weighted importance sampling estimator* (WIS), as it forms a weighted average of returns G_{t_k}. The weighted importance sampling estimator $V_n^{\text{WIS}}(s)$ is defined as

$$V_n^{\text{WIS}}(s) \stackrel{\text{def}}{=\!=} \frac{\sum_{k=1}^n W_{t_k} G_{t_k}}{\sum_{k=1}^n W_{t_k}}. \tag{4.4}$$

Both OIS and WIS are consistent, whereas only OIS is an unbiased estimator. We formally state these properties in the following. For simplicity, we assume the trajectories are generated i.i.d.

Theorem 2 (Unbiasedness of OIS)**.** *Let Assumption 1 hold. If the trajectories* $\{L_{t_k}\}_{k=1}^n$ *are generated i.i.d. from state* s *and* $\{W_{t_k}\}_{k=1}^n$ *are corresponding importance weights, then* $V_n^{\mathrm{OIS}}(s)$ *is an unbiased estimator of* $v_\pi(s)$*, that is:*

$$\mathrm{E}_\mu\left[V_n^{\mathrm{OIS}}(s)\right] = v_\pi(s). \tag{4.5}$$

Proof. With $X_k \overset{\text{def}}{=\!=} W_{t_k} G_{t_k}$, $V_n^{\mathrm{OIS}}(s)$ is an average estimator. Therefore, the result follows from Lemmas 1 and 9. □

Theorem 3 (Bias of WIS)**.** *Let Assumption 1 hold. If the trajectories* $\{L_{t_k}\}_{k=1}^n$ *are generated i.i.d. from state* s *and* $\{W_{t_k}\}_{k=1}^n$ *are corresponding importance weights, then* $V_n^{\mathrm{OIS}}(s)$ *is a biased estimator of* $v_\pi(s)$*, that is, generally:*

$$\mathrm{E}_\mu\left[V_n^{\mathrm{WIS}}(s)\right] \neq v_\pi(s). \tag{4.6}$$

Proof. It can be easily shown by taking $n = 1$. Then we can write:

$$\mathrm{E}_\mu\left[V_n^{\mathrm{WIS}}(s)\right] = \mathrm{E}_\mu\left[\frac{W_{t_1} G_{t_1}}{W_{t_1}}\middle| S_{t_1} = s\right] = \mathrm{E}_\mu\left[G_{t_1} | S_{t_1} = s\right] = v_\mu(s) \neq v_\pi(s), \text{ as } \mu \neq \pi. \tag{4.7}$$

□

Theorem 4 (Consistency of OIS)**.** *Let Assumption 1 hold. If the trajectories* $\{L_{t_k}\}_{k=1}^n$ *are generated i.i.d. from state* s *and* $\{W_{t_k}\}_{k=1}^n$ *are corresponding importance weights, then* $V_n^{\mathrm{OIS}}(s)$ *is a consistent estimator of* $v_\pi(s)$*, that is:*

$$V_n^{\mathrm{OIS}}(s) \xrightarrow{\text{a.s.}} v_\pi(s). \tag{4.8}$$

Proof. With $X_k \overset{\text{def}}{=\!=} W_{t_k} G_{t_k}$, $V_n^{\mathrm{OIS}}(s)$ is an average estimator. Therefore, the result follows from Lemmas 3 and 9. □

Theorem 5 (Consistency of WIS)**.** *Let Assumption 1 hold. If the trajectories* $\{L_{t_k}\}_{k=1}^n$ *are generated i.i.d. from state* s *and* $\{W_{t_k}\}_{k=1}^n$ *are corresponding importance weights, then* $V_n^{\mathrm{WIS}}(s)$ *is a consistent estimator of* $v_\pi(s)$*, that is:*

$$V_n^{\mathrm{WIS}}(s) \xrightarrow{\text{a.s.}} v_\pi(s). \tag{4.9}$$

Proof. With $X_k \overset{\text{def}}{=\!=} W_{t_k} G_{t_k}$ and $Y_k \overset{\text{def}}{=\!=} W_{t_k}$, $V_n^{\mathrm{OIS}}(s)$, is a ratio estimator. Therefore, the result follows from Lemmas 4, 5 and 9. □

The relationship between the EMSE of WIS and the MSE of OIS can be established using (2.32):

$$\text{EMSE}\left[V_n^{\text{WIS}}(s)\right] < \text{MSE}[V_n^{\text{OIS}}(s)] \tag{4.10}$$

$$\implies \text{EMSE}\left[\frac{\bar{X}_n}{\bar{Y}_n}\right] < \text{MSE}[\bar{X}_n] \tag{4.11}$$

$$\implies \mu_x \,\text{Cov}[X_1, Y_1] > \frac{\mu_x^2 \,\text{Var}[Y_1]}{2} \tag{4.12}$$

$$\implies v_\pi(s)\,\text{Cov}_\mu[W_t, W_t G_t | S_t = s] > \frac{1}{2} v_\pi(s)^2 \,\text{Var}_\mu[W_t | S_t = s]. \tag{4.13}$$

The covariance term can be simplified in the following way:

$$\text{Cov}_\mu[W_t, W_t G_t | S_t = s] = \text{E}_\mu\left[(W_t - \text{E}_\mu[W_t|S_t=s])(W_t G_t - \text{E}_\mu[W_t G_t|S_t=s])|S_t = s\right] \tag{4.14}$$

$$= \text{E}_\mu\left[(W_t - 1)(W_t G_t - v_\pi(s))|S_t = s\right] \tag{4.15}$$

$$= \text{E}_\mu\left[W_t(W_t G_t - v_\pi(s))|S_t = s\right] - \text{E}_\mu\left[W_t G_t - v_\pi(s)|S_t = s\right] \tag{4.16}$$

$$= \text{E}_\pi\left[W_t G_t - v_\pi(s)|S_t = s\right] \tag{4.17}$$

$$= \text{E}_\pi\left[W_t G_t|S_t = s\right] - v_\pi(s) \tag{4.18}$$

$$= \text{E}_\pi\left[W_t G_t|S_t = s\right] - \text{E}_\pi\left[W_t|S_t = s\right] v_\pi(s) + \text{E}_\pi\left[W_t|S_t = s\right] v_\pi(s) - v_\pi(s) \tag{4.19}$$

$$= \text{Cov}_\pi[W_t, G_t|S_t = s] + v_\pi(s)\left(\text{E}_\pi\left[W_t|S_t = s\right] - 1\right) \tag{4.20}$$

$$= \text{Cov}_\pi[W_t, G_t|S_t = s] + v_\pi(s)\left(\text{E}_\mu\left[W_t^2|S_t = s\right] - \text{E}_\mu\left[W_t|S_t = s\right]^2\right) \tag{4.21}$$

$$= \text{Cov}_\pi[W_t, G_t|S_t = s] + v_\pi(s)\,\text{Var}_\mu[W_t|S_t = s]. \tag{4.22}$$

Therefore, the EMSE of WIS is smaller than the MSE of OIS, when:

$$v_\pi(s)\,\text{Cov}_\mu[W_t, W_t G_t|S_t = s] > \frac{1}{2} v_\pi(s)^2 \,\text{Var}_\mu[W_t|S_t = s] \tag{4.23}$$

$$\implies v_\pi(s)\left(\text{Cov}_\pi[W_t, G_t|S_t = s] + v_\pi(s)\,\text{Var}_\mu[W_t|S_t = s]\right) > \frac{1}{2} v_\pi(s)^2 \,\text{Var}_\mu[W_t|S_t = s] \tag{4.24}$$

$$\implies v_\pi(s)\,\text{Cov}_\pi[W_t, G_t|S_t = s] > -\frac{1}{2} v_\pi(s)^2 \,\text{Var}_\mu[W_t|S_t = s] \tag{4.25}$$

$$\implies \begin{cases} |v_\pi(s)|\,\text{Cov}_\pi[W_t, G_t|S_t = s] > -\frac{1}{2} v_\pi(s)^2 \,\text{Var}_\mu[W_t|S_t = s]; \text{ if } v_\pi(s) \geq 0 \\ |v_\pi(s)|\,\text{Cov}_\pi[W_t, G_t|S_t = s] < \frac{1}{2} v_\pi(s)^2 \,\text{Var}_\mu[W_t|S_t = s]; \text{ if } v_\pi(s) < 0. \end{cases} \tag{4.26}$$

This gives a simpler characterization of the case when WIS is likely to have lower MSE than OIS. The right-hand side of the inequality above is negative. Therefore, if the importance weight and the return are even loosely correlated under the target policy and the value $v_\pi(s)$ is non-negative, then WIS is much likely to have lesser MSE than OIS. If all the trajectories are equally likely under the behavior policy, then this covariance is easier to interpret. In that case, if a trajectory that is more likely under the target policy is also more likely to produce larger returns, then this covariance becomes higher. In many

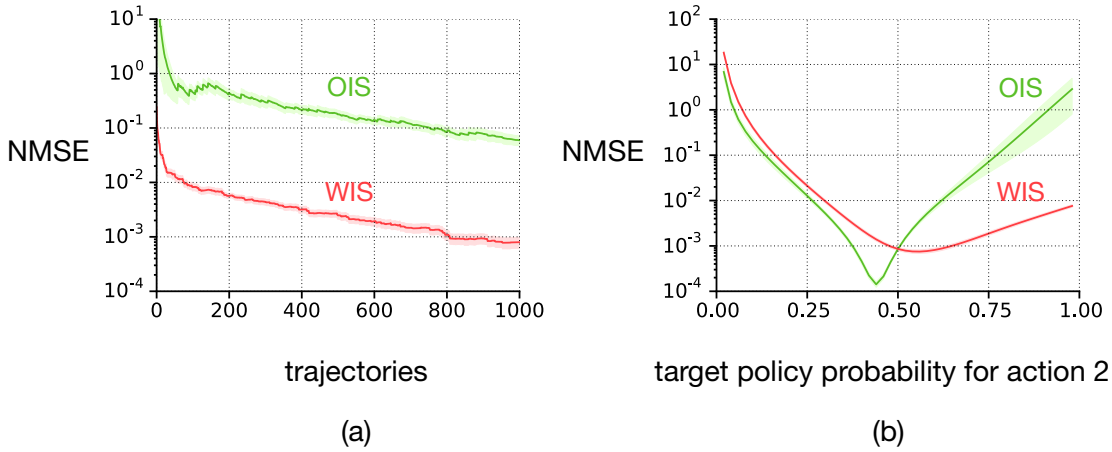

Figure 4.1: Performance comparison of OIS and WIS estimators: *Left*: In an off-policy task with a simple Markov chain, WIS performed better than OIS by more than an order of magnitude margin. *Right*: WIS is preferable to OIS for a certain range of target policy probabilities for a particular action. In this problem, the higher the probability of choosing that action under the target policy, the larger is the covariance between the importance weights and the returns. The result here confirms our analytical finding that WIS is preferable when the importance weights and the returns have positive covariance.

off-policy problems, target policies are chosen in such a way that they are more likely to produce larger returns, such as greedy policies in control problems. This partially explains why WIS tends to perform better than OIS in practice. Moreover, the larger the variance of the importance weights, the more likely for WIS to obtain lesser MSE than OIS. However, negative state values are not as much favorable for WIS.

The following experiments illustrate the comparison between WIS and OIS on an idealized prediction task. This task consisted a chain-like MDP with a state space $\mathcal{S} = \{1, 2, \cdots, 10\}$. Two actions 1 and 2 were available at every state. For states $\{1, 2, \cdots, 9\}$, both actions increased the state by one. At state 10, both actions brought the agent to state 1. The behavior policy chose both actions uniformly randomly.

The general value function was defined in the following way. The pseudo-reward was $+1$ for action 2 and 0 for action 1 with standard normal noise of zero mean and 0.1 standard deviation added in both cases. The state-dependent termination was 0 for state 10 and 0.99 for the rest of the states. The target policy chose action 2 with probability 0.9 at every state.

We evaluated both OIS and WIS on this task by generating 1000 trajectories each starting from state 1 and ending at state 10 following the behavior policy. The performance was measured by the estimated value V of state 1 subtracted from the true value v, squared and then normalized by the square of the true value: $\frac{(v-V)^2}{v^2}$. The results were averaged over 50 independent runs for statistical significance. We refer to this measure as the Normalized Mean Squared Error (NMSE).

The left panel of Figure 4.1 shows NMSE of OIS and WIS over 1000 trajectories. WIS performed better than OIS in this task by more than an order of magnitude during the full period of learning. The right panel of Figure 4.1 shows NMSE of OIS and WIS for evaluating target policies with different probabilities $p(2)$ of choosing action 2, ranging from 0.02 to 0.98 in steps of 0.02. The importance weights and returns under the target policy are positively correlated for $p(2) > 0.5$ and are negatively correlated for $p(2) < 0.5$. In this task, the value is always non-negative for all values of $p(2)$. WIS performed better than OIS up to two orders of magnitude when $p(2) > 0.5$. On the other hand, WIS performed worse than OIS when $p(2) < 0.5$ but not by more than an order of magnitude.

4.2 Discounting-aware Off-policy Estimators

The scaling of the return G_t with the importance weight W_t of the complete trajectory has a potential disadvantage. The trajectory corresponding to the return can be arbitrarily large, whereas the discounting involved with the return may make the impact of the later part of the trajectory on the return insignificant. However, the full importance weight W_t is not aware of such discounting. It will scale a lightly discounted return the same way it will scale a heavily discounted return. For example, consider a full trajectory of 100 steps long, that is $\gamma_{100} = 0$, where each of the rewards from step 10 is heavily discounted with $(\gamma_k)_{k=10}^{99} = 0.001$. The return corresponding to this trajectory effectively depends on the rewards from the first 10 steps. In this case, scaling the return with the importance weight W_1^{100} of the full trajectory may introduce unnecessary variance. In general, the longer the trajectory is, the higher the variance of the importance weight can be.

One way we can devise a discounting-aware importance weighting is by applying importance weights separately to separate undiscounted or flat return components of the complete return. A return can be expressed in terms of the weighted sum of different multistep flat returns where each flat return is weighted by the degree of discounting applied to it. Let us define a *flat return* starting from time step t up to horizon h as

$$\bar{G}_t^h = R_{t+1} + R_{t+2} + \cdots + R_h. \tag{4.27}$$

Then the complete return G_t can be written in terms of the flat returns with different horizons in the following way:

$$G_t = \sum_{l=t+1}^{T(t)} \prod_{i=t+1}^{l-1} \gamma_i R_l \tag{4.28}$$

$$= \sum_{l=t+1}^{T(t)} R_l \left(\prod_{i=t+1}^{l-1} \gamma_i - \underbrace{\prod_{i=t+1}^{T(t)} \gamma_i}_{0} \right) \tag{4.29}$$

$$= \sum_{l=t+1}^{T(t)} R_l \left(\left(\prod_{i=t+1}^{l-1} \gamma_i - \prod_{i=t+1}^{l} \gamma_i \right) + \left(\prod_{i=t+1}^{l} \gamma_i - \prod_{i=t+1}^{l+1} \gamma_i \right) + \cdots + \left(\prod_{i=t+1}^{T(t)-1} \gamma_i - \prod_{i=t+1}^{T(t)} \gamma_i \right) \right) \tag{4.30}$$

$$= \sum_{l=t+1}^{T(t)} R_l \sum_{h=l}^{T(t)} \left(\prod_{i=t+1}^{h-1} \gamma_i - \prod_{i=t+1}^{h} \gamma_i \right) \tag{4.31}$$

$$= \sum_{l=t+1}^{T(t)} R_l \sum_{h=l}^{T(t)} (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \tag{4.32}$$

$$= \sum_{l=t+1}^{T(t)} \sum_{h=l}^{T(t)} R_l \, (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \tag{4.33}$$

$$= \sum_{h=t+1}^{T(t)} (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \sum_{l=t+1}^{h} R_l \tag{4.34}$$

$$= \sum_{h=t+1}^{T(t)} (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \bar{G}_t^h. \tag{4.35}$$

It can be viewed as each flat return $\bar{G}_t^h$ being weighted by its probability of termination $(1-\gamma_h)\prod_{i=t+1}^{h-1}\gamma_i$ at the horizon h.

Then, instead of scaling the return with the complete importance weight W_t, we can weight each individual flat returns $\bar{G}_t^h$ with the importance weights W_t^h of their corresponding trajectories. We denote this new scaled return by $\tilde{G}_t$, which is defined as follows:

$$\tilde{G}_t = \sum_{h=t+1}^{T(t)} W_t^h \, (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \bar{G}_t^h. \tag{4.36}$$

In the following, we show that $\tilde{G}_t$ is an unbiased estimate of $v_\pi(s)$.

Lemma 10 (Unbiasedness of flat-return-specific scaling)**.** *Let Assumption 1 hold. If the state visited at time t is s, then $\tilde{G}_t$ defined by (4.36) is an unbiased estimator of $v_\pi(s)$, that is:*

$$\mathrm{E}_\mu \left[\tilde{G}_t | S_t = s \right] = v_\pi(s). \tag{4.37}$$

Proof. By expanding $\tilde{G}_t$, we can write:

$$\mathrm{E}_\mu \left[\tilde{G}_t | S_t = s \right] = \mathrm{E}_\mu \left[\sum_{h=t+1}^{T(t)} W_t^h \, (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \sum_{l=t+1}^{h} R_l | S_t = s \right] \tag{4.38}$$

$$= \sum_{h=t+1}^{\infty} \mathrm{E}_\mu \left[W_t^h \, (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \sum_{l=t+1}^{h} R_l | S_t = s \right] \tag{4.39}$$

$$= \sum_{h=t+1}^{\infty} \mathrm{E}_\pi \left[(1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \sum_{l=t+1}^{h} R_l | S_t = s \right] \tag{4.40}$$

$$= \mathrm{E}_\pi \left[\sum_{h=t+1}^{T(t)} (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \sum_{l=t+1}^{h} R_l | S_t = s \right] \tag{4.41}$$

$$= \mathrm{E}_\pi \left[G_t | S_t = s \right] \tag{4.42}$$

$$= v_\pi(s). \tag{4.43}$$

□

Using $X_k \stackrel{\text{def}}{=\!=} \tilde{G}_{t_k}$, we can form an average estimator. We consider it as an improvement over OIS, and thus call it *OIS2*. It is defined as follows:

$$V_n^{\text{OIS2}}(s) \stackrel{\text{def}}{=\!=} \frac{\sum_{k=1}^{n} \tilde{G}_{t_k}}{n}. \tag{4.44}$$

We can form a ratio estimator using $X_k \stackrel{\text{def}}{=\!=} \tilde{G}_{t_k}$ as well. The question is what we should use for Y_k, where $\mathrm{E}_\mu[Y_k | S_{t_k} = s] = 1$. We could use the full importance weight W_{t_k}, but we are not using this in the numerator. We can use the same reasoning based on the awareness of discounting we used for the scaled return $\tilde{G}_{t_k}$ to form an importance weight to be used in the denominator. We denote it by $\tilde{W}_{t_k}$ and define it as follows:

$$\tilde{W}_{t_k} = \sum_{h=t_k+1}^{T_k} W_{t_k}^h \, (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i. \tag{4.45}$$

The expected value of $\tilde{W}_{t_k}$ is 1, which we formally show in the following.

Lemma 11 (New importance weight)**.** *Given Assumption 1, the following holds:*

$$\mathrm{E}_\mu \left[\tilde{W}_t \middle| S_t = s \right] = 1. \tag{4.46}$$

Proof. Using the definition of expectations, we can write:

$$\mathrm{E}_\mu \left[\tilde{W}_t \middle| S_t = s \right] = \mathrm{E}_\mu \left[\sum_{h=t+1}^{T(t)} W_t^h \, (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \middle| S_t = s \right] \tag{4.47}$$

$$= \sum_{h=t+1}^{\infty} \mathrm{E}_\mu \left[W_t^h \, (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \middle| S_t = s \right] \tag{4.48}$$

$$= \sum_{h=t+1}^{\infty} \mathrm{E}_\pi \left[(1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \middle| S_t = s \right] \tag{4.49}$$

$$= \mathrm{E}_\pi \left[\sum_{h=t+1}^{T(t)} (1-\gamma_h) \prod_{i=t+1}^{h-1} \gamma_i \middle| S_t = s \right] \tag{4.50}$$

$$= 1. \tag{4.51}$$

□

We call this ratio estimator *WIS2*. It is defined as follows:

$$V_n^{\mathrm{WIS2}}(s) \stackrel{\mathrm{def}}{=\!=} \frac{\sum_{k=1}^{n} \tilde{G}_{t_k}}{\sum_{k=1}^{n} \tilde{W}_{t_k}}. \tag{4.52}$$

Both OIS2 and WIS2 are consistent, whereas only OIS2 is an unbiased estimator. We formally state these properties in the following.

Theorem 6 (Unbiasedness of OIS2)**.** *Let Assumption 1 hold and the trajectories* $\{L_{t_k}\}_{k=1}^{n}$ *are generated i.i.d. from state* s*. If* $\{\tilde{G}_{t_k}\}_{k=1}^{n}$ *are defined by (4.36), then* $V_n^{\mathrm{OIS2}}(s)$ *is an unbiased estimator of* $v_\pi(s)$*, that is:*

$$\mathrm{E}_\mu\left[V_n^{\mathrm{OIS2}}(s)\right] = v_\pi(s). \tag{4.53}$$

Proof. With $X_k \stackrel{\mathrm{def}}{=\!=} \tilde{G}_{t_k}$, $V_n^{\mathrm{OIS2}}(s)$ is an average estimator. Therefore, the result follows from Lemmas 1 and 10. □

Theorem 7 (Bias of WIS2)**.** *Let Assumption 1 hold and the trajectories* $\{L_{t_k}\}_{k=1}^{n}$ *are generated i.i.d. from state* s*. If* $\{\tilde{G}_{t_k}\}_{k=1}^{n}$ *are defined by (4.36) and* $\{\tilde{W}_{t_k}\}_{k=1}^{n}$ *by (4.45), then* $V_n^{\mathrm{WIS2}}(s)$ *is a biased estimator of* $v_\pi(s)$*, that is:*

$$\mathrm{E}_\mu\left[V_n^{\mathrm{WIS2}}(s)\right] \neq v_\pi(s). \tag{4.54}$$

Proof. We prove it using a counterexample. Consider an MDP where discounting is 1 everywhere except at the state where full termination occurs. In that case, WIS2 is equivalent to WIS. The proof of Theorem 3 gives a simple counterexmple where WIS is not unbiased, which applies to WIS2 in the current example as well. Therefore, it follows that WIS2 is a biased estimator of $v_\pi(s)$.

□

Theorem 8 (Consistency of OIS2)**.** *Let Assumption 1 hold and the trajectories* $\{L_{t_k}\}_{k=1}^{n}$ *are generated i.i.d. from state* s*. If* $\{\tilde{G}_{t_k}\}_{k=1}^{n}$ *are defined by (4.36), then* $V_n^{\mathrm{OIS2}}(s)$ *is a consistent estimator of* $v_\pi(s)$*, that is:*

$$\mathrm{E}_\mu\left[V_n^{\mathrm{OIS2}}(s)\right] \xrightarrow{\text{a.s.}} v_\pi(s). \tag{4.55}$$

Proof. With $X_k \stackrel{\mathrm{def}}{=\!=} \tilde{G}_{t_k}$, $V_n^{\mathrm{OIS2}}(s)$ is an average estimator. Therefore, the result follows from Lemmas 3 and 10. □

Theorem 9 (Consistency of WIS2)**.** *Let Assumption 1 hold and the trajectories* $\{L_{t_k}\}_{k=1}^n$ *are generated i.i.d. from state* s*. If* $\{\tilde{G}_{t_k}\}_{k=1}^n$ *are defined by (4.36) and* $\{\tilde{W}_{t_k}\}_{k=1}^n$ *by (4.45), then* $V_n^{\mathrm{WIS2}}(s)$ *is a consistent estimator of* $v_\pi(s)$*, that is:*

$$\mathrm{E}_\mu\left[V_n^{\mathrm{WIS2}}(s)\right] \xrightarrow{\text{a.s.}} v_\pi(s). \tag{4.56}$$

Proof. With $X_k \overset{\text{def}}{=} \tilde{G}_{t_k}$ and $Y_k \overset{\text{def}}{=} \tilde{W}_{t_k}$, $V_n^{\mathrm{WIS2}}(s)$, is a ratio estimator. Therefore, the result follows from Lemmas 4, 10 and 11. □

The MSE of WIS2 can be compared with that of OIS2 the similar way as we have done it for WIS and OIS:

$$\mathrm{EMSE}\left[V_n^{WIS2}(s)\right] < \mathrm{MSE}[V_n^{OIS2}(s)] \tag{4.57}$$

$$\implies \mathrm{EMSE}\left[\frac{\bar{X}_n}{\bar{Y}_n}\right] < \mathrm{MSE}[\bar{X}_n] \tag{4.58}$$

$$\implies \mu_x \,\mathrm{Cov}[X_1, Y_1] > \frac{\mu_x^2 \,\mathrm{Var}[Y_1]}{2} \tag{4.59}$$

$$\implies v_\pi(s)\,\mathrm{Cov}_\mu[\tilde{W}_t, \tilde{G}_t | S_t = s] > \frac{1}{2} v_\pi(s)^2\,\mathrm{Var}_\mu[\tilde{W}_t | S_t = s]. \tag{4.60}$$

The covariance term can be simplified in the following way:

$$\mathrm{Cov}_\mu[\tilde{W}_t, \tilde{G}_t | S_t = s] = \mathrm{E}_\mu\left[(\tilde{W}_t - \mathrm{E}_\mu[\tilde{W}_t | S_t = s])(\tilde{G}_t - \mathrm{E}_\mu[\tilde{G}_t | S_t = s]) | S_t = s\right] \tag{4.61}$$

$$= \mathrm{E}_\mu\left[(\tilde{W}_t - 1)(\tilde{G}_t - v_\pi(s)) | S_t = s\right] \tag{4.62}$$

$$= \mathrm{E}_\mu\left[\tilde{W}_t(\tilde{G}_t - v_\pi(s)) | S_t = s\right] - \mathrm{E}_\mu\left[\tilde{G}_t - v_\pi(s) | S_t = s\right] \tag{4.63}$$

$$= \mathrm{E}_\pi\left[\tilde{W}_t G_t - v_\pi(s) | S_t = s\right] \tag{4.64}$$

$$= \mathrm{E}_\pi\left[\tilde{W}_t G_t | S_t = s\right] - \mathrm{E}_\pi\left[\tilde{W}_t | S_t = s\right] v_\pi(s) + \mathrm{E}_\pi\left[\tilde{W}_t | S_t = s\right] v_\pi(s) - v_\pi(s) \tag{4.65}$$

$$= \mathrm{Cov}_\pi[\tilde{W}_t, G_t | S_t = s] + v_\pi(s)\left(\mathrm{E}_\mu\left[\tilde{W}_t^2 | S_t = s\right] - \mathrm{E}_\mu\left[\tilde{W}_t | S_t = s\right]^2\right) \tag{4.66}$$

$$= \mathrm{Cov}_\pi[\tilde{W}_t, G_t | S_t = s] + v_\pi(s)\,\mathrm{Var}_\mu[\tilde{W}_t | S_t = s]. \tag{4.67}$$

Therefore, the EMSE of WIS2 is smaller than the MSE of OIS2, when:

$$v_\pi(s)\,\mathrm{Cov}_\mu[\tilde{W}_t, G_t | S_t = s] > \frac{1}{2} v_\pi(s)^2\,\mathrm{Var}_\mu[\tilde{W}_t | S_t = s] \tag{4.68}$$

$$\implies v_\pi(s)\left(\mathrm{Cov}_\pi[\tilde{W}_t, G_t | S_t = s] + v_\pi(s)\,\mathrm{Var}_\mu[\tilde{W}_t | S_t = s]\right) > \frac{1}{2} v_\pi(s)^2\,\mathrm{Var}_\mu[\tilde{W}_t | S_t = s] \tag{4.69}$$

$$\implies v_\pi(s)\,\mathrm{Cov}_\pi[\tilde{W}_t, G_t | S_t = s] > -\frac{1}{2} v_\pi(s)^2\,\mathrm{Var}_\mu[\tilde{W}_t | S_t = s] \tag{4.70}$$

$$\implies \begin{cases} |v_\pi(s)|\,\mathrm{Cov}_\pi[\tilde{W}_t, G_t | S_t = s] > -\frac{1}{2} v_\pi(s)^2\,\mathrm{Var}_\mu[\tilde{W}_t | S_t = s]; \text{ if } v_\pi(s) \geq 0 \\ |v_\pi(s)|\,\mathrm{Cov}_\pi[\tilde{W}_t, G_t | S_t = s] < \frac{1}{2} v_\pi(s)^2\,\mathrm{Var}_\mu[\tilde{W}_t | S_t = s]; \text{ if } v_\pi(s) < 0. \end{cases} \tag{4.71}$$

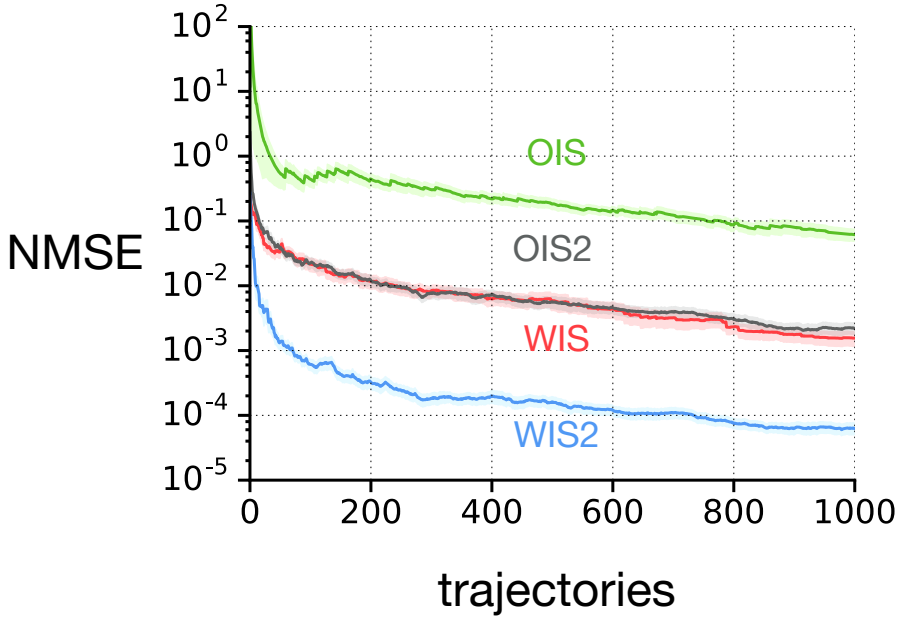

Figure 4.2: The new off-policy tabular estimators OIS2 and WIS2 perform substantially better than OIS and WIS, respectively, in off-policy tasks with variable discounting.

This characterization of MSE of WIS2 and OIS2 is similar to that of WIS and OIS. But rather than depending on the correlation of the return and the importance weights of the full trajectory, this depends on the correlation of the return and the importance weights of the effective portion of the trajectory. However, this does not show how WIS2 compares with WIS.

To demonstrate a case where OIS2 and WIS2 are preferable to WIS and OIS, respectively, we modify the previous task. In the modified task, the trajectories start at state 1 and end in state 10, but the returns are heavily discounted after state 2. More specifically, the state-dependent discounting is 1 for state 1 and 2, and 0.1 from state 3 to 9. The target policy probability for choosing action 2 is still 0.9. Figure 4.2 shows the performance of OIS2 and WIS2 in comparison with OIS and WIS. Both OIS2 and WIS2 performed better than OIS and WIS, respectively, as expected. Moreover, the performance of OIS2 was similar to that of WIS almost for the full period of learning.

4.3 Reward-Specific Off-policy Estimators

There is yet another way of applying the importance weights to account for the discrepancy between the target and the behavior policies. Instead of scaling full returns or different multi-step flat-returns, we may choose to scale each reward individually so that in expectation the reward appears to be generated by the target policy. In this case, the new scaled

return is defined in the following way:

$$\hat{G}_t = \sum_{l=t+1}^{T(t)} W_t^l \prod_{i=t+1}^{l-1} \gamma_i R_l. \tag{4.72}$$

Here each reward R_l is scaled by the importance weight W_t^l corresponding to the trajectory starting from time t when state s is visited up to the time the reward was generated.

In the following, we show that $\hat{G}_t$ is an unbiased estimate of $v_\pi(s)$.

Lemma 12 (Unbiasedness of reward-specific scaling)**.** *Let Assumption 1 hold. If the state visited at time t is s, then $\hat{G}_t$ defined by (4.72) is an unbiased estimator of $v_\pi(s)$, that is:*

$$\mathrm{E}_\mu \left[\hat{G}_t | S_t = s \right] = v_\pi(s). \tag{4.73}$$

Proof. By expanding $\hat{G}_t$, we can write:

$$\begin{aligned}
\mathrm{E}_\mu \left[\hat{G}_t | S_t = s \right] &= \mathrm{E}_\mu \left[\sum_{l=t+1}^{T(t)} W_t^l \prod_{i=t+1}^{l-1} \gamma_i R_l | S_t = s \right] && (4.74)\\
&= \sum_{l=t+1}^{\infty} \mathrm{E}_\mu \left[W_t^l \prod_{i=t+1}^{l-1} \gamma_i R_l | S_t = s \right] && (4.75)\\
&= \sum_{l=t+1}^{\infty} \mathrm{E}_\pi \left[\prod_{i=t+1}^{l-1} \gamma_i R_l | S_t = s \right] && (4.76)\\
&= \mathrm{E}_\pi \left[\sum_{l=t+1}^{T(t)} \prod_{i=t+1}^{l-1} \gamma_i R_l | S_t = s \right] && (4.77)\\
&= \mathrm{E}_\pi \left[G_t | S_t = s \right] && (4.78)\\
&= v_\pi(s). && (4.79)
\end{aligned}$$

□

Using $X_k \stackrel{\text{def}}{=\!=} \hat{G}_{t_k}$, we can form an average estimator, which we call the *per-reward importance sampling estimator* (PRIS), can be defined in the following way:

$$V_n^{\text{PRIS}}(s) \stackrel{\text{def}}{=\!=} \frac{\sum_{k=1}^n \hat{G}_{t_k}}{n}. \tag{4.80}$$

In the following, we show the statistical properties of PRIS is similar to the other average estimators.

Theorem 10 (Unbiasedness of PRIS)**.** *Let Assumption 1 hold and the trajectories* $\{L_{t_k}\}_{k=1}^n$ *are generated i.i.d. from state* s*. If* $\{\hat{G}_{t_k}\}_{k=1}^n$ *are defined by (4.72), then* $V_n^{\mathrm{PRIS}}(s)$ *is an unbiased estimator of* $v_\pi(s)$*, that is:*

$$\mathrm{E}_\mu\left[V_n^{\mathrm{PRIS}}(s)\right] = v_\pi(s). \tag{4.81}$$

Proof. With $X_k \stackrel{\mathrm{def}}{=\!=} \hat{G}_{t_k}$, $V_n^{\mathrm{PRIS}}(s)$ is an average estimator. Therefore, the result follows from Lemmas 1 and 12. □

Theorem 11 (Consistency of PRIS)**.** *Let Assumption 1 hold and the trajectories* $\{L_{t_k}\}_{k=1}^n$ *are generated i.i.d. from state* s*. If* $\{\hat{G}_{t_k}\}_{k=1}^n$ *are defined by (4.72), then* $V_n^{\mathrm{PRIS}}(s)$ *is a consistent estimator of* $v_\pi(s)$*, that is:*

$$\mathrm{E}_\mu\left[V_n^{\mathrm{PRIS}}(s)\right] \xrightarrow{\text{a.s.}} v_\pi(s). \tag{4.82}$$

Proof. With $X_k \stackrel{\mathrm{def}}{=\!=} \hat{G}_{t_k}$, $V_n^{\mathrm{PRIS}}(s)$ is an average estimator. Therefore, the result follows from Lemmas 3 and 12. □

PRIS was proposed by Precup et al. (2000), which they called *per-decision importance sampling estimator*. Precup et al. intended to develop an estimator based on PRIS that is similar to what WIS is to OIS. They called this estimator *weighted per-decision importance sampling (WPDIS) estimator*, which is defined as follows:

$$V_n^{\mathrm{WPDIS}}(s) = \frac{\sum_{k=1}^n \hat{G}_{t_k}}{\sum_{k=1}^n \hat{\hat{W}}_{t_k}}, \tag{4.83}$$

$$\hat{\hat{W}}_{t_k} = \sum_{l=t_k}^{T_k} W_{t_k}^{l+1} \prod_{i=t_k+1}^{l} \gamma_i. \tag{4.84}$$

WPDIS is another ratio estimator, but it is different than the above ratio estimators in the sense that the summands of the denominator are not in expectation one, which is the main step to make sure that the ratio estimator is a consistent estimator. Thomas (2015) showed that PDW is not a consistent estimator.

Thomas (2015) introduced another estimator based on PRIS, which is called *consistent weighted per-decision importance sampling* (CWPDIS) estimator. We call this *WPRIS*. Thomas introduced this estimator together with the normalization of the returns using the upper and lower bounds of returns. All the estimators introduced here can be extended using such normalized returns, In general, such bounds may not be known, and we avoid this modification for simplicity.

WPRIS is not a ratio estimator but is a discounted sum of a series of ratio estimators and is introduced for constant discounting. Consider that n trajectories have been observed, where H is the length of the longest trajectory. Then WPRIS forms H ratio estimators,

where the lth estimator is a weighted average of the lth rewards from each trajectory, and the corresponding importance weights are the weights of the average. The trajectories smaller than the longest trajectories are padded by pretending that the rest of the transitions have occurred with the same discounting and zero rewards to and from an absorbing state that allows a single action. The following is the definition of the WPRIS estimator:

$$V_n^{\text{WPRIS}}(s) \stackrel{\text{def}}{=\!=} \sum_{l=1}^{H} \gamma^{l-1} \frac{\sum_{k=1}^{n} W_{t_k}^{t_k+l} R_{t_k+l}}{\sum_{k=1}^{n} W_{t_k}^{t_k+l}}. \tag{4.85}$$

Each of the H weighted averages above are a consistent estimator of the lth reward. Therefore, $V_n^{\text{WPRIS}}(s)$ is a consistent estimator of $v_\pi(s)$ with fixed horizon H and constant discounting γ. Ther is no available extension of WPRIS to state-dependent discounting, and the main difficulty is due to having different discounting for rewards from different trajectories.

PRIS and WPRIS may potentially reduce variance as each reward is scaled by the minimal amount of importance weight for correcting the discrepancy between policies. However, if MDP is such that the rewards tend to cancel each other, then scaling the returns directly is likely to produce less variance than scaling the rewards individually.

In the following four experiments, we illustrate cases where PRIS and WPRIS perform better and worse compared to OIS2 and WIS2, respectively. In the first experiment, we used the same off-policy evaluation task where we evaluated OIS and WIS. The top-left plot of Figure 4.3 shows the results. In this experiment, PRIS and WPRIS performed better than OIS2 and WIS2, respectively. The performance of WPRIS was about an order of magnitude better than that of WIS2.

In the second experiment, we used a modification of the previous task. Here, rewards for all transitions are -1 except the transition from state 9 to 10, where the reward is 11. The top-right plot of Figure 4.3 shows the result. In this experiment, PRIS performed considerably worse than OIS2. The advantage of using WPRIS over WIS2 is also reduced.

WPRIS has a significant drawback compared to WIS2 in that rather than working with different returns it aligns different rewards occurring at the same position of transitions across different trajectories and averages them. Therefore, if the trajectories are vastly different from each other, then the rewards are not likely to be similar after the same number of transitions. We cannot see this drawback in the above two experiments because the trajectories are extremely similar to each other with only white noise added to each transition.

Our third and fourth experiments emphasize this drawback of WPRIS. In these two experiments, we used an MDP where the agent may randomly follow different trajectories that are vastly different from each other. In this MDP, there are now 11 states $\{1, \cdots, 11\}$ forming a chain. From each state, there are two actions available -1 and 1. From states 2 to 10, choosing action -1 decreases the state index and choosing action 1 increases it. In the third experiment, there is a reward of +1 for choosing action 1 and a reward of -1

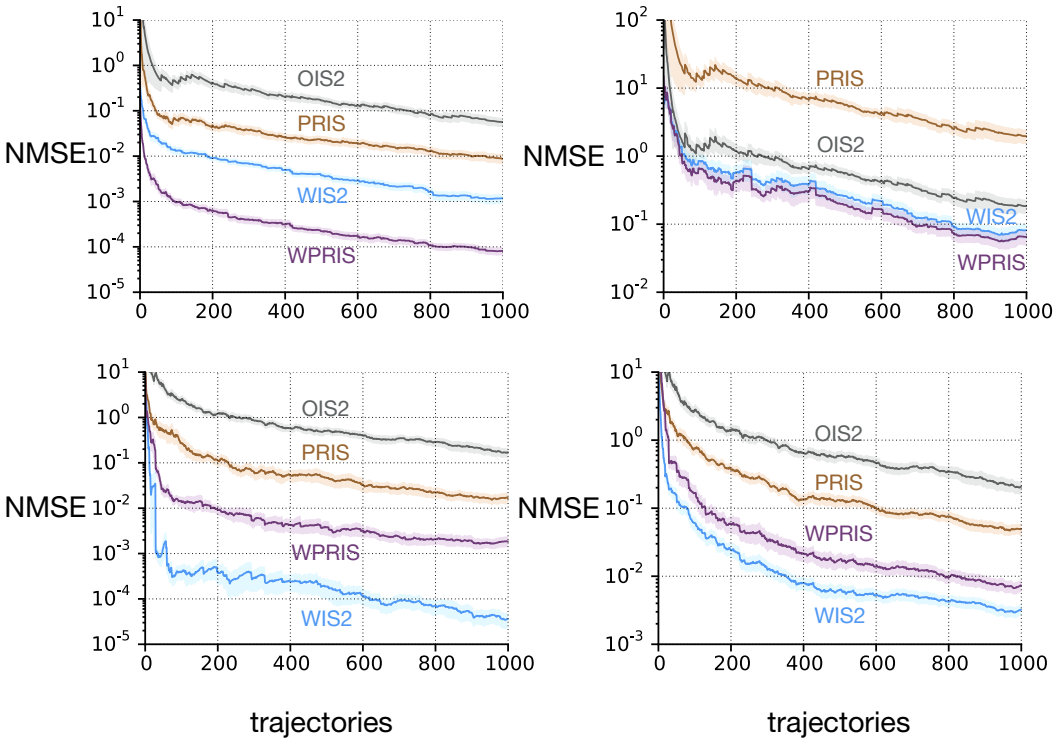

Figure 4.3: Performance comparison between reward specific estimators (PRIS and WPRIS) and the new discounting aware estimators (OIS2 and WIS2). *Upper Left*: In a simple off-policy task with fixed length trajectories, the reward specific estimators performed substantially better than the corresponding discounting-aware estimators. *Upper Right*: However, the performance advantage of the reward-specific estimators is diminished when the rewards are canceling each other along the trajectory. *Lower Left*: the ratio-based reward-specific estimator WPRIS performed considerably worse than the ratio-based discounting-aware estimator WIS2, when the length of the trajectories are allowed to be variable, while the rewards are still canceling each other along the trajectory. *Lower Right*: When the trajectories are of variable length, WPRIS still performed worse compared to WIS2 even when the rewards did not cancel along the trajectory.

for choosing action -1 from those states. In the fourth experiment, there is a reward of +1 for choosing action 1 and a reward of 0 for choosing action -1 from those states. In states 1 and 11, both actions lead to the middle state 6 with reward 0. The behavior policy chooses both actions uniformly randomly whereas the target policy chooses action 1 with probability 0.9. The bottom-left and bottom-right plots of Figure 4.3 shows the results of the third and the fourth experiments, respectively. In both of these experiments, WPRIS performed worse than WIS2. In the third experiment where the rewards are likely to cancel along a trajectory, WPRIS performed more than by an order of magnitude than WIS2.

4.4 Incremental Updates of Off-policy Estimators

In this section, we focus on the incremental implementation of the tabular off-policy estimators both on a per-trajectory and real-time basis. We developed the notion of per-trajectory learning setting and per-trajectory incrementality in Chapter 2. In that Chapter,

All the estimators, except WPRIS, are either an average estimator or a ratio estimator. In both cases, deriving a per-trajectory incremental implementation is straightforward. In these estimators, the index n stands for the number of samples as well as trajectories. Therefore, the goal here is to express each estimator indexed by n in terms of the same estimator indexed by $n-1$ and in terms of the data arriving only from nth trajectory.

As discussed in Chapter 3, the average estimator $\bar{X}_n$ defined by (2.3) can be implemented per-trajectory incrementally in two different ways. It can be implemented incrementally by updating the sum incrementally in the numerator:

$$\hat{X}_n = \sum_{k=1}^{n} X_k = \hat{X}_{n-1} + X_n, \tag{4.86}$$

$$\bar{X}_n = \frac{\hat{X}_n}{n}. \tag{4.87}$$

The average estimator can be also be updated per-trajectory incrementally with a direct recursion:

$$\bar{X}_n = \frac{\sum_{k=1}^{n} X_k}{n} = \bar{X}_{n-1} + \frac{1}{n}\left(X_n - \bar{X}_n\right). \tag{4.88}$$

On the other hand, the ratio estimator is the ratio of two summations:

$$\bar{R}_n = \frac{\hat{X}_n}{\hat{Y}_n}. \tag{4.89}$$

The ratio estimator can be implemented incrementally by updating the sums of numerator and the denominator incrementally. Let us define $\hat{X}_n \stackrel{\text{def}}{=\!=} \sum_{k=1}^{n} X_k$ and $\hat{Y}_n \stackrel{\text{def}}{=\!=} \sum_{k=1}^{n} Y_k$. These are simple summations and can be incrementally updated in the following way:

$$\hat{X}_n = \sum_{k=1}^{n} X_k = \hat{X}_{n-1} + X_n, \tag{4.90}$$

$$\hat{Y}_n = \sum_{k=1}^{n} Y_k = \hat{Y}_{n-1} + Y_n. \tag{4.91}$$

An alternative incremental implementation depends on the definition of the ratio estimator as the ratio of two averages: $\bar{R}_n \stackrel{\text{def}}{=\!=} \frac{\bar{X}_n}{\bar{Y}_n}$ given by (2.4). Then the ratio estimator can be implemented incrementally by updating both the numerator and the denominator incrementally:

$$\bar{X}_n = \bar{X}_{n-1} + \frac{1}{n}\left(X_n - \bar{X}_n\right), \tag{4.92}$$

$$\bar{Y}_n = \bar{Y}_{n-1} + \frac{1}{n}\left(Y_n - \bar{Y}_n\right), \tag{4.93}$$

$$\bar{R}_n = \frac{\bar{X}_n}{\bar{Y}_n}. \tag{4.94}$$

Although the above two implementations are incremental, they are not recursive on the original estimator $\bar{R}_n$. However, it is also possible to derive an incremental implementation of the ratio estimator $\bar{R}_n$ that is also recursive on $\bar{R}_n$. In the following we show how to achieve that:

$$\bar{R}_n = \frac{\bar{X}_n}{\bar{Y}_n} = \frac{\hat{X}_n}{\hat{Y}_n} \tag{4.95}$$

$$= \frac{\hat{X}_{n-1} + X_n}{\hat{Y}_n} \tag{4.96}$$

$$= \frac{\hat{Y}_{n-1}}{\hat{Y}_n} \times \frac{\hat{X}_{n-1}}{\hat{Y}_{n-1}} + \frac{X_n}{\hat{Y}_n} \tag{4.97}$$

$$= \frac{\hat{Y}_n - Y_n}{\hat{Y}_n} \times \bar{R}_{n-1} + \frac{X_n}{\hat{Y}_n} \tag{4.98}$$

$$= \left(1 - \frac{Y_n}{\hat{Y}_n}\right) \times \bar{R}_{n-1} + \frac{X_n}{\hat{Y}_n} \tag{4.99}$$

$$= \bar{R}_{n-1} + \frac{1}{\hat{Y}_n}\left(X_n - Y_n \bar{R}_{n-1}\right). \tag{4.100}$$

A slight modification to this update is where $\hat{Y}_n$ is replaced by the equivalent term $n\bar{Y}_n$:

$$\bar{R}_n = \bar{R}_{n-1} + \frac{1}{n\bar{Y}_n}\left(X_n - Y_n \bar{R}_{n-1}\right). \tag{4.101}$$

We have given two equivalent implementations of the average estimator and four equivalent implementations of the ratio estimator. Although they are equivalent, they are not numerically equally stable. Previous works showed that the direct recursive implementation of the average estimator defined by (4.88) is numerically more stable than the implementation with the recursion of the summation in the numerator defined by (4.86) and (4.87) (van Reeken 1968, 1970, Ling 1974). Similarly, it can be shown that the implementation of the ratio estimator based on the recursion of sum defined by (4.89) and the direct recursion defined by (4.100) that still uses a sum in its implementation are numerically less stable than the other two implementations of the ratio estimator (4.94) and (4.101). The latter two are more stable prominently because they avoid implementations of sums, which may grow to a quantity much larger than the quantity being estimated resulting in loss of precision.

The WPRIS estimator is neither an average estimator nor a ratio estimator. It is a summation of a number of ratio estimators. When a new trajectory is complete, the corresponding new terms are distributed among the ratio estimators that are summands of the outer summation of WPRIS. These terms corresponding to the newest trajectory

appear both in the numerator and the denominator of each of the ratio estimator. It can be seen by observing the definition of both $V_n^{\text{WPRIS}}(s)$ and $V_{n+1}^{\text{WPRIS}}(s)$:

$$V_n^{\text{WPRIS}}(s) = \sum_{l=1}^{H} \gamma^{l-1} \frac{\sum_{k=1}^{n} W_{t_k}^{t_k+l} R_{t_k+l}}{\sum_{k=1}^{n} W_{t_k}^{t_k+l}} \tag{4.102}$$

$$V_{n+1}^{\text{WPRIS}}(s) = \sum_{l=1}^{H} \gamma^{l-1} \frac{\sum_{k=1}^{n+1} W_{t_k}^{t_k+l} R_{t_k+l}}{\sum_{k=1}^{n+1} W_{t_k}^{t_k+l}} \tag{4.103}$$

$$= \sum_{l=1}^{H} \gamma^{l-1} \frac{\sum_{k=1}^{n} W_{t_k}^{t_k+l} R_{t_k+l} + W_{t_{n+1}}^{t_{n+1}+l} R_{t_{n+1}+l}}{\sum_{k=1}^{n} W_{t_k}^{t_k+l} + W_{t_{n+1}}^{t_{n+1}+l}}. \tag{4.104}$$

The second terms both in the numerator and the denominator of the last equation is what get appended to the previous update of WPRIS estimator.

In order to develop a per-trajectory incremental implementation of WPRIS, we first re-write $V_n^{\text{WPRIS}}(s)$ as follows:

$$V_n^{\text{WPRIS}}(s) = \sum_{l=1}^{H} \gamma^{l-1} \frac{\sum_{k=1}^{n} W_{t_k}^{t_k+l} R_{t_k+l}}{\sum_{k=1}^{n} W_{t_k}^{t_k+l}} \tag{4.105}$$

$$= \sum_{l=1}^{H} \gamma^{l-1} \frac{P_n(l)}{Q_n(l)}, \tag{4.106}$$

$$P_{n+1}(c) = \sum_{k=1}^{n+1} W_{t_k}^{t_k+c} R_{t_k+c} = \sum_{k=1}^{n} W_{t_k}^{t_k+c} R_{t_k+c} + W_{t_{n+1}}^{t_{n+1}+c} R_{t_{n+1}+c} \tag{4.107}$$

$$= P_n(c) + W_{t_{n+1}}^{t_{n+1}+c} R_{t_{n+1}+c}, \tag{4.108}$$

$$Q_{n+1}(c) = \sum_{k=1}^{n+1} W_{t_k}^{t_k+c} = \sum_{k=1}^{n} W_{t_k}^{t_k+c} + W_{t_{n+1}}^{t_{n+1}+c} \tag{4.109}$$

$$= Q_n(c) + W_{t_{n+1}}^{t_{n+1}+c}. \tag{4.110}$$

Then $V_{n+1}^{\text{WPRIS}}(s)$ can be written as:

$$V_{n+1}^{\text{WPRIS}}(s) = \sum_{l=1}^{H} \gamma^{l-1} \frac{P_{n+1}(l)}{Q_{n+1}(l)} = \sum_{l=1}^{H} \gamma^{l-1} \frac{P_n(l) + W_{t_{n+1}}^{t_{n+1}+l} R_{t_{n+1}+l}}{Q_n(l) + W_{t_{n+1}}^{t_{n+1}+l}}. \tag{4.111}$$

This update requires storing the sequences $(P_n(l))_{l=1}^n$ and $(Q_n(l))_{l=1}^n$, both occupying $O(H)$ memory. The terms from the newest trajectory have to be incorporated with these two sequences to obtain $V_{n+1}^{\text{WPRIS}}(s)$, and the computation of which requires $O(H)$ operations. Is it possible to implement WPRIS in a per-trajectory strictly incremental manner? As we have shown in Chapter 3, to bring the summation over samples outside, the ones inside P and Q here, algebraic expressions of this form require a distributivity law that does not hold. It appears unlikely that WPRIS can be implemented strictly incrementally.

Real-time strictly incremental implementation is possible for all the estimators presented here except WPRIS. We extend some of these estimators to the case of function

approximation and bootstrapping in the subsequent chapters and provide the incremental implementations of them there. Therefore, we do not repeat the incremental implementation for the special case of tabular representation here. For WPRIS, it is not possible to implement a real-time strictly incremental update. It can be easily seen from the fact that there is always a possibility that an unseen trajectory can be longer than any of the trajectories seen so far. Therefore, the maximum length of all observed trajectories H can increase as time step increases. As WPRIS requires maintaining $O(H)$ ratio estimators, its memory can always increase with more time steps. Therefore, it cannot be strictly incremental.

4.5 Discussion and Conclusions

In this section, we discussed and analyzed six tabular off-policy estimators. We analyzed the bias, variance, and convergence of these estimators. There are three main groups among these six estimators, each containing a pair of estimators. These groups differ from each other based on how the importance weights are applied. They are discussed in Sections 4.1, 4.2 and 4.3. Each pair consists of an average estimator and a ratio estimator. Categorizing the off-policy estimators in terms of these standard forms simplified the analysis considerably and opened up opportunities to develop new estimators more easily. Among the six estimators, two of them are new.

We included experiments illustrating how these estimators differ from each other and what cases make one estimator perform better than the others. At least for two of the groups, we were able to characterize theoretically the cases where the ratio estimators can perform better than the average estimators. This characterization also provides a better understanding on why historically importance sampling with ratio estimators performed better than importance sampling with the average estimators. The pseudo-reward signals can be scaled to satisfy one condition or the other so that we can choose among these estimators in a principled way. In our experiments, it was observed that when ratio estimators perform better, the performance margin is much larger than when the average estimators perform better.

From all the experiments, it appears that among the ratio estimators, one of the new estimators WIS2 is preferable to WIS when state-dependent discounting is used. WPRIS is likely to perform better than other ratio estimators when the trajectories are similar. However, WPRIS is by design incompatible with real-time incremental learning. In a more natural setting, WIS2 performed better than WPRIS.

Chapter 5

Weighted Importance Sampling with Function Approximation[1]

In this chapter, we introduce a novel and principled way of extending weighted importance sampling to parametric function approximation, which constitutes a foundational step toward developing a practically applicable off-policy algorithm based on weighted importance sampling. As we have found that WIS2 is generally preferable among all the other importance sampling methods, we adopt it as the core element for extending to function approximation. By extending WIS2 to linear function approximation, we develop WIS2-LS, one of the main contributions of this thesis. However, due to its simplicity, we first start with WIS for extending.

5.1 WIS as Weighted Least Squares

In this section, we devise a systematic way of extending tabular algorithms to linear function approximation. Extending average estimators is straightforward and will be discussed first here. Trickier is ratio estimators such as WIS, for which a backward-consistent extension was not available. By drawing the intuition from average estimators, we devise a systematic way of extending WIS to linear function approximation, which would eventually lead us to extend WIS2 later in this chapter.

An average estimator can be seen as the solution to the least-squares problem. If we are estimating v, and $\{X_k\}_{k=1}^n$ are unbiased i.i.d. samples representative of v, then a reasonable empirical objective function is as follows:

$$J_n(\hat{v}) \stackrel{\text{def}}{=\!=} \frac{1}{n}\sum_{k=1}^{n}(X_k - \hat{v})^2 \,. \tag{5.1}$$

Justification behind choosing this objective function is that $J_n(\hat{v})$ is an unbiased and consistent estimate of the Mean Squared Error (MSE): $\mathrm{E}[(X_k - \hat{v})^2]$. If V is an estimator

[1]The core concept of this chapter is developed in a published paper coauthored by this author (Mahmood, van Hasselt & Sutton 2014).

that minimizes this function, then we can easily see that V must be the average estimator:

$$\frac{\partial}{\partial V}\frac{1}{n}\sum_{k=1}^{n}(X_k - V)^2 = 0 \implies -\frac{2}{n}\sum_{k=1}^{n}(X_k - V) = 0 \implies V = \frac{1}{n}\sum_{k=1}^{n} X_k = \bar{X}_n. \tag{5.2}$$

In the off-policy general value function estimation problem, our main objective is more complicated due the the discrepancy between policies. In particular, we would like to estimate $v_\pi(s) = E_\pi[G_k|S_k = s]$ under policy π, where we have a sequence of trajectories $\{L_k\}_{k=1}^n$ generated by following policy μ. Then the MSE is defined by $\mathrm{E}_\pi[(G_k - \hat{v})^2|S_k = s]$. Two empirical objective functions we can naturally think of based on this MSE are as follows:

$$\tilde{J}_n(\hat{v}) = \frac{1}{n}\sum_{k=1}^{n}\left(W_k G_k - \hat{v}\right)^2, \tag{5.3}$$

$$\hat{J}_n(\hat{v}) = \frac{1}{n}\sum_{k=1}^{n} W_k\left(G_k - \hat{v}\right)^2, \tag{5.4}$$

where W_k is the importance weight of G_k. Here, the first one is in the least-squares form, but is neither an unbiased nor a consistent estimate of MSE. On the other hand, the second one is a *weighted* least-squares form and is both an unbiased and consistent estimate of MSE.

Interestingly, we notice that OIS is the solution to the first objective whereas WIS is the solution to the second:

$$\frac{\partial}{\partial V}\tilde{J}_n(V) = 0 \implies -\frac{2}{n}\sum_{k=1}^{n}(W_k G_k - V) = 0 \implies V = \frac{1}{n}\sum_{k=1}^{n} W_k G_k = V_n^{\mathrm{OIS}}, \tag{5.5}$$

$$\frac{\partial}{\partial V}\hat{J}_n(V) = 0 \implies -\frac{2}{n}\sum_{k=1}^{n} W_k\left(G_k - V\right) = 0 \implies V = \frac{\sum_{k=1}^{n} W_k G_k}{\sum_{k=1}^{n} W_k} = V_n^{\mathrm{WIS}}. \tag{5.6}$$

For off-policy tasks, WIS appears to be more directly related to a more meaningful objective function than OIS. In Chapter 4, we have already shown that both of these estimators are consistent. The fact that WIS is a solution to a weighted least-squares problem is the key observation here that enables us to extend WIS to parametric function approximation.

5.2 WIS with Linear Function Approximation

In this section, we use the fact that WIS is a solution to a form of weighted least-squares problem and apply it to develop a new method for linear function approximation. The aim is to develop a method that carries over the benefits of WIS in this extended setting and at the same time maintains backward consistency, meaning that when the function approximation degenerates to the tabular setting, this new method becomes exactly equivalent to WIS.

We use the same linear function approximation setting as described in Section 2.4. Two objective functions related to this setting are the MSE:

$$\mathrm{E}_d\left[\left(v_\pi(S_k) - \boldsymbol{\theta}^\top \boldsymbol{\phi}_k\right)^2\right] = \sum_s d(s)\left(v_\pi(s) - \boldsymbol{\theta}^\top \boldsymbol{\phi}(s)\right)^2, \tag{5.7}$$

and the Mean Squared Return Error (MSRE):

$$\mathrm{E}_{\pi,d}\left[\left(G_k - \boldsymbol{\theta}^\top \boldsymbol{\phi}_k\right)^2\right] = \mathrm{E}_d\left[\mathrm{E}_\pi\left[\left(G_k - \boldsymbol{\theta}^\top \boldsymbol{\phi}_k\right)^2 | S_k = s\right]\right]. \tag{5.8}$$

Here, $d(s)$ denotes the probability for choosing state s. Both objective functions have the same solution, they are related to each other in the following way: MSRE = MSE + $\mathrm{Var}_{\pi,d}[G_k]$.

We may wonder what the proper state distribution d should be for our objective function. It did not matter in the tabular case because the solution is not affected by the state distribution; the solution for each state could be estimated separately from those of the others. However, in the function approximation case as the approximation resource, namely the features, are shared among the states, the solution potentially depends on how the states are distributed.

In supervised learning, a similar problem occurs where the distribution of the input affects the solution, resulting in model misspecification. The problem exacerbates when the desired state distribution differs from the state distribution of the data. In the supervised learning scenario with model misspecification where the input distribution of the training data is different than that of the test data, this problem is called *the covariate shift problem* (Shimodaira 2000).

One may speculate that as the problem at hand is about estimating the value while following the target policy π, this policy should also influence how the states themselves are visited. Therefore, one may desire a solution where the discrepancy of not only the return distribution is corrected but that of the state distribution is also corrected. While this is an important goal to fulfill, we view the off-policy learning problem separately from the covariate shift problem and focus mainly on correcting the discrepancy of the return distribution. As we only consider the off-policy learning problem in this thesis, our desired MSRE is $\mathrm{E}_{\pi,d_\mu}[(G_k - \boldsymbol{\theta}^\top \boldsymbol{\phi}_k)^2]$, where d_μ denotes the stationary state distribution in data, which is induced by the behavior policy μ. The solution $\boldsymbol{\theta}_*$ to this objective function is as follows:

$$\boldsymbol{\theta}_* \stackrel{\text{def}}{=\!=} \arg\min_{\boldsymbol{\theta}} \mathrm{E}_{\pi,d_\mu}\left[(G_k - \boldsymbol{\theta}^\top \boldsymbol{\phi}_k)^2\right] \tag{5.9}$$

$$= \mathrm{E}_{d_\mu}\left[\boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top\right]^{-1} \mathrm{E}_{\pi,d_\mu}\left[G_k \boldsymbol{\phi}_k\right] \tag{5.10}$$

$$= \mathrm{E}_{d_\mu}\left[\boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top\right]^{-1} \mathrm{E}_{d_\mu}\left[v_\pi(S_k)\boldsymbol{\phi}_k\right]. \tag{5.11}$$

Here, we assume that $\mathrm{E}_{d_\mu}\left[\boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top\right]$ is nonsingular.

Similar to the tabular setting discussed in Section 5.1, we provide two empirical objective functions associated with MSRE:

$$\tilde{J}_t(\boldsymbol{\theta}) \stackrel{\text{def}}{=\!=} \frac{1}{n}\sum_{k=1}^{n}\left(W_{t_k}G_{t_k} - \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t_k}\right)^2, \tag{5.12}$$

$$\hat{J}_t(\boldsymbol{\theta}) \stackrel{\text{def}}{=\!=} \frac{1}{n}\sum_{k=1}^{n} W_{t_k}\left(G_{t_k} - \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t_k}\right)^2. \tag{5.13}$$

These two empirical objectives are generalizations of the two tabular empirical objectives (5.3), and (5.4) to linear function approximation. The second empirical objective function is an unbiased and consistent estimator of MSRE. By solving the above empirical objectives, we obtain the following two estimators, respectively:

$$\tilde{\boldsymbol{\theta}}_n \stackrel{\text{def}}{=\!=} \left(\frac{1}{n}\sum_{k=1}^{n}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top\right)^{-1}\left(\frac{1}{n}\sum_{k=1}^{n}W_{t_k}G_{t_k}\boldsymbol{\phi}_{t_k}\right), \tag{5.14}$$

$$\hat{\boldsymbol{\theta}}_n \stackrel{\text{def}}{=\!=} \left(\frac{1}{n}\sum_{k=1}^{n}W_{t_k}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top\right)^{-1}\left(\frac{1}{n}\sum_{k=1}^{n}W_{t_k}G_{t_k}\boldsymbol{\phi}_{t_k}\right). \tag{5.15}$$

The first estimator is a backward compatible extension to OIS and the second one to WIS. We call $\tilde{\boldsymbol{\theta}}$ the *OIS-LS estimator* and $\hat{\boldsymbol{\theta}}$ the *WIS-LS estimator*. In the following theorems, we formally prove these facts and some of the key properties regarding these two estimators. These properties are similar to some of those of the tabular estimators we investigated in Chapter 4, showing that they are proper extensions of their corresponding tabular estimators.

Theorem 12 (Unbiasedness of OIS-LS)**.** *If v_π is a linear function of the features, that is, $v_\pi(s) = {\boldsymbol{\theta}_*}^\top \boldsymbol{\phi}(s)$, then OIS-LS is an unbiased estimator, that is, $\mathrm{E}_{\mu,d_\mu}[\tilde{\boldsymbol{\theta}}_n] = \boldsymbol{\theta}_*$.*

Proof. The proof is given by the following derivation:

$$\begin{aligned}
\mathrm{E}_{\mu,d_\mu}[\tilde{\boldsymbol{\theta}}_n] &= \mathrm{E}_{\mu,d_\mu}\left[\left(\frac{1}{n}\sum_{k=1}^{n}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top\right)^{-1}\left(\frac{1}{n}\sum_{k=1}^{n}W_{t_k}G_{t_k}\boldsymbol{\phi}_{t_k}\right)\right] \\
&= \mathrm{E}_{d_\mu}\left[\left(\frac{1}{n}\sum_{k=1}^{n}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top\right)^{-1}\left(\frac{1}{n}\sum_{k=1}^{n}\mathrm{E}_\mu\left[W_{t_k}G_{t_k}|S_{t_k}\right]\boldsymbol{\phi}_{t_k}\right)\right] \\
&= \mathrm{E}_{d_\mu}\left[\left(\frac{1}{n}\sum_{k=1}^{n}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top\right)^{-1}\left(\frac{1}{n}\sum_{k=1}^{n}\mathrm{E}_\pi\left[G_{t_k}|S_{t_k}\right]\boldsymbol{\phi}_{t_k}\right)\right] \\
&= \mathrm{E}_{d_\mu}\left[\left(\frac{1}{n}\sum_{k=1}^{n}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top\right)^{-1}\left(\frac{1}{n}\sum_{k=1}^{n}v_\pi(S_{t_k})\boldsymbol{\phi}_{t_k}\right)\right]
\end{aligned}$$

$$= \mathrm{E}_{d_\mu}\left[\left(\frac{1}{n}\sum_{k=1}^{n}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top\right)^{-1}\left(\frac{1}{n}\sum_{k=1}^{n}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top\boldsymbol{\theta}_*\right)\right]$$
$$= \mathrm{E}_{d_\mu}\left[\left(\frac{1}{n}\sum_{k=1}^{n}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top\right)^{-1}\left(\frac{1}{n}\sum_{k=1}^{n}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top\right)\right]\boldsymbol{\theta}_* = \boldsymbol{\theta}_*.$$

□

Theorem 13 (Bias of WIS-LS)**.** *Even if v_π is a linear function of the features, that is, $v_\pi(s) = \boldsymbol{\theta}_*^\top\boldsymbol{\phi}(s)$, WIS-LS is in general a biased estimator, that is, $\mathrm{E}_{\mu,d_\mu}[\hat{\boldsymbol{\theta}}_n] \neq \boldsymbol{\theta}_*$.*

Proof. We prove it by providing a counterexample to the claim that $\mathrm{E}_{\mu,d_\mu}[\hat{\boldsymbol{\theta}}_n] = \boldsymbol{\theta}_*$. Consider $\mathcal{S} = \{s\}$ and $\boldsymbol{\phi}(s) = [1]$. It is easy to see that in this case $\boldsymbol{\theta}_* = \mathrm{E}_\pi[G_k|s] = v_\pi(s)$. The WIS-LS estimator $\hat{\boldsymbol{\theta}}_n$ also reduces to the WIS estimator:

$$\hat{\boldsymbol{\theta}}_n = \left(\frac{1}{n}\sum_{k=1}^{n}W_{t_k}\right)^{-1}\left(\frac{1}{n}\sum_{k=1}^{n}W_{t_k}G_{t_k}\right) = V_n^{\mathrm{WIS}},$$

which is a biased estimator of $v_\pi(s)$. Hence, in general, $\mathrm{E}_{\mu,d_\mu}[\hat{\boldsymbol{\theta}}_n] \neq \boldsymbol{\theta}_*$. □

Theorem 14 (Consistency of OIS-LS)**.** *The OIS-LS estimator $\tilde{\boldsymbol{\theta}}_n$ is a consistent estimator of the MSE solution $\boldsymbol{\theta}_*$ given in (5.11).*

Proof. Due to the strong law of large numbers

$$\frac{1}{n}\sum_{k=1}^{n}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top \xrightarrow{\text{a.s.}} \mathrm{E}_{d_\mu}\left[\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top\right],$$
$$\frac{1}{n}\sum_{k=1}^{n}W_{t_k}G_{t_k}\boldsymbol{\phi}_{t_k} \xrightarrow{\text{a.s.}} \mathrm{E}_{\mu,d_\mu}\left[W_kG_k\boldsymbol{\phi}_k\right] = \mathrm{E}_{d_\mu}\left[\mathrm{E}_\mu\left[W_kG_k|S_k\right]\boldsymbol{\phi}_k\right] = \mathrm{E}_{d_\mu}\left[v_\pi(S_k)\boldsymbol{\phi}_k\right].$$

Then it follows that $\tilde{\boldsymbol{\theta}}_n \xrightarrow{\text{a.s.}} \boldsymbol{\theta}_*$. □

Theorem 15 (Consistency of WIS-LS)**.** *The WIS-LS estimator $\hat{\boldsymbol{\theta}}_n$ is a consistent estimator of the MSE solution $\boldsymbol{\theta}_*$ given in (5.11).*

Proof. Due to the strong law of large numbers

$$\frac{1}{n}\sum_{k=1}^{n}W_{t_k}\boldsymbol{\phi}_{t_k}\boldsymbol{\phi}_{t_k}^\top \xrightarrow{\text{a.s.}} \mathrm{E}_{\mu,d_\mu}\left[W_k\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top\right] = \mathrm{E}_{d_\mu}\left[\mathrm{E}_\mu\left[W_k|S_k\right]\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top\right] = \mathrm{E}_{d_\mu}\left[\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top\right],$$

$$\frac{1}{n}\sum_{k=1}^{n} W_{t_k} G_{t_k} \boldsymbol{\phi}_{t_k} \xrightarrow{\text{a.s.}} \mathrm{E}_{\mu,d_\mu}\left[W_k G_k \boldsymbol{\phi}_k\right] = \mathrm{E}_{d_\mu}\left[\mathrm{E}_\mu\left[W_k G_k | S_k\right] \boldsymbol{\phi}_k\right] = \mathrm{E}_{d_\mu}\left[v_\pi(S_k)\boldsymbol{\phi}_k\right].$$

Then it follows that $\hat{\boldsymbol{\theta}}_n \xrightarrow{\text{a.s.}} \boldsymbol{\theta}_*$. □

Theorem 16 (Backward compatibility of OIS-LS with OIS)**.** *If the features form an orthonormal basis, then the OIS-LS estimate $\tilde{\boldsymbol{\theta}}_n^\top \boldsymbol{\phi}(s)$ is equivalent to the OIS estimate of $v_\pi(s)$.*

Proof. Let $\boldsymbol{\Phi}$ denote to be the feature matrix the rows of which contain the feature vectors of different unique inputs: $\boldsymbol{\Phi} = \left(\boldsymbol{\phi}(s_1), \ldots, \boldsymbol{\phi}(s_{|\mathcal{S}|})\right)^\top$, where $s_1, \ldots, s_{|\mathcal{S}|}$ are different unique states. Then the vector containing the estimated value of each unique state according to the OIS-LS estimator can be written as

$$\boldsymbol{\Phi}\tilde{\boldsymbol{\theta}}_n = \boldsymbol{\Phi}\left(\sum_{s\in\mathcal{S}} n_s \boldsymbol{\phi}(s)\boldsymbol{\phi}(s)^\top\right)^{-1} \sum_{s\in\mathcal{S}}\left(\sum_{i=1}^{n_s} W_{s,i} G_{s,i}\right)\boldsymbol{\phi}(s) = \boldsymbol{\Phi}\left(\boldsymbol{\Phi}^\top \mathbf{N}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^\top \mathbf{y},$$

where n_s is the number of times state s has been the start state among n samples, $G_{s,i}$ is the return corresponding to the ith occurrence of state s as a start state and $W_{s,i}$ is the corresponding importance weight. Here, $\mathbf{N}$ is a diagonal matrix where the ith diagonal element contains n_{s_i}: $\mathbf{N} = \mathrm{diag}\left(n_{s_1}, \ldots, n_{s_{|\mathcal{S}|}}\right)$ and $\mathbf{y} = \left(\sum_{i=1}^{n_{s_1}} W_{s_1,i} G_{s_1,i}, \ldots, \sum_{i=1}^{n_{|\mathcal{S}|}} W_{|\mathcal{S}|,i} G_{|\mathcal{S}|,i}\right)^\top$.

Note that, due to orthonormality of the features, $\boldsymbol{\Phi}$ is necessarily a square matrix and full rank. Therefore, it follows that the vector of the estimates can be written as

$$\boldsymbol{\Phi}\tilde{\boldsymbol{\theta}}_n = \boldsymbol{\Phi}\boldsymbol{\Phi}^{-1}\mathbf{N}^{-1}\boldsymbol{\Phi}^{\top -1}\boldsymbol{\Phi}^\top \mathbf{y} = \mathbf{N}^{-1}\mathbf{y}.$$

The element of this vector corresponding to any state s is the ordinary importance-sampling estimator of its corresponding value: $n_s^{-1}\sum_{i=1}^{n_s} W_{s,i} G_{s,i}$. □

Theorem 17 (Backward compatibility of WIS-LS with WIS)**.** *If the features form an orthonormal basis, then the WIS-LS estimate $\hat{\boldsymbol{\theta}}_n^\top \boldsymbol{\phi}(s)$ of state s is equivalent to the WIS estimate of the value of s.*

Proof. The proof is similar to the proof of Theorem 5. First, we write the vector of the estimates according to the WIS-LS estimate as

$$\boldsymbol{\Phi}\hat{\boldsymbol{\theta}}_n = \boldsymbol{\Phi}\left(\sum_{s\in\mathcal{S}}\left(\sum_{i=1}^{n_s} W_{s,i}\right)\boldsymbol{\phi}(s)\boldsymbol{\phi}(s)^\top\right)^{-1}\sum_{s\in\mathcal{S}}\left(\sum_{i=1}^{n_s}\rho_{s,i} Y_{s,i}\right)\boldsymbol{\phi}(s) = \boldsymbol{\Phi}\left(\boldsymbol{\Phi}^\top \mathbf{R}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^\top \mathbf{y},$$

where $\mathbf{R}$ is a diagonal matrix with each diagonal element containing the total summation of the importance weights corresponding to each state:

$$\mathbf{R} = \text{diag}\left(\left(\sum_{i=1}^{n_{s_1}} W_{s_1,i}\right), \ldots, \left(\sum_{i=1}^{n_{s_{|\mathcal{S}|}}} W_{s_{|\mathcal{S}|},i}\right)\right).$$

Hence, the vector of estimates can be written as

$$\mathbf{\Phi}\hat{\boldsymbol{\theta}}_n = \mathbf{\Phi}\mathbf{\Phi}^{-1}\mathbf{R}^{-1}\mathbf{\Phi}^{\top -1}\mathbf{\Phi}^{\top}\mathbf{y} = \mathbf{R}^{-1}\mathbf{y},$$

The element of this vector corresponding to any state s is the WIS estimate of its corresponding outputs: $\left(\sum_{i=1}^{n_s} W_{s,i}\right)^{-1} \sum_{i=1}^{n_s} W_{s,i} G_{s,i}$. □

We test these newly developed algorithms analogous to OIS and WIS first on a simple toy task and then on the Mountain Car task.

In the toy task, we adopted a supervised learning setting. In this task, there are 25 inputs $\{s_1, \cdots, s_{25}\}$ representing states, and from each input, there is a different distribution of outputs. The output can take 100 different values $\{0, 1, \cdots, 99\}$ according to a given distribution plus a noise according to normal distribution with 0.1 standard deviation. Two distributions were generated by choosing probabilities using random numbers drawn uniformly randomly and normalized so that they add up to 1. One is the target distribution under which we wish to estimate the value, and the other is the sampling distribution which was used to generate data. Two different sets of feature vectors were generated. For each input, 10 features were generated and kept fixed. For the first set, the features were either 0 or 1 generated uniformly randomly, and for the second set, the features were between $[0, 1]$ generated using uniform random distribution.

Data consisting 1000 outputs $\{G_k\}_{k=1}^{1000}$ were generated, each time choosing one of the inputs uniformly randomly and then using the sampling distribution to generate the output corresponding to that input. For each of these outputs G_k, the importance weight W_k was generated by dividing the target distribution probability of that output by its the sampling distribution probability. Using the 1000 triples of outputs G_k, importance weights W_k and input feature vectors ϕ_k, OIS-LS estimator $\tilde{\boldsymbol{\theta}}$ and WIS-LS estimator $\hat{\boldsymbol{\theta}}$ were calculated. As the matrices $\frac{1}{n}\sum_{k=1}^{n} \boldsymbol{\phi}_k\boldsymbol{\phi}_k^{\top}$ and $\frac{1}{n}\sum_{k=1}^{n} W_k\boldsymbol{\phi}_k\boldsymbol{\phi}_k^{\top}$ may be singular, especially at the beginning, a regularization of $(10^{-4}/n)\mathbf{I}$ was added to them for numerical stability. Each time a triple is presented, the estimators are computed and their performance is measured as $\sum_{i=1}^{25}\left(\boldsymbol{\theta}_*^{\top}\boldsymbol{\phi}(s_i) - \boldsymbol{\theta}^{\top}\boldsymbol{\phi}(s_i)\right)^2$, where $\boldsymbol{\theta}$ is either the OIS-LS estimator or the WIS-LS estimator and $\boldsymbol{\theta}_*$ is the MSE solution according to (5.11). We call this the Mean Squared Projected Error (MSPE). This performance measure was generated for 1000 samples and then averaged over 30 such different independent sets of 1000 samples. It was repeated for the two different sets of feature vectors. The MSPE is then normalized by $\sum_{i=1}^{25}\left(\boldsymbol{\theta}_*^{\top}\boldsymbol{\phi}(s_i)\right)^2$

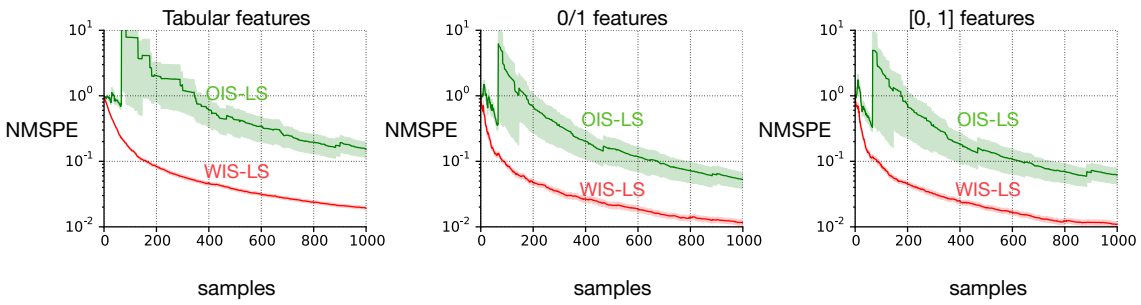

Figure 5.1: Performance comparison between OIS-LS and WIS-LS on a toy supervised learning task with three different sets of features. Performance of both estimators is shown in Normalized Mean Squared Projected Error (NMSPE) for learning over 1000 samples, averaged over 30 independently generated data sets.

so that the unit error can be used as a baseline performance. We refer to this measure as the Normalized MSPE (NMSPE).

In Figure 5.1, we show the performance of WIS-LS and OIS-LS in NMSPE. The middle plot corresponds to 0/1 features, and the right plot corresponds to $[0, 1]$ features. In order to make sure that the problem is favorable to tabular WIS, we show the performance of WIS-LS and OIS-LS with tabular features, in which case they reduce to WIS and OIS respectively. In both cases of function approximation, WIS-LS substantially outperformed OIS-LS throughout the period of learning.

For the Mountain Car off-policy prediction task, we used the standard simulator as described by Sutton and Barto (1998). We used Sarsa(0.9) with replacing traces and ϵ-greedy policy with $\epsilon = 0.01$ to learn a control policy for 2000 episodes. This learned policy served as the target policy. Then we estimated the value of 25 different random start states under the learned policy by generating 50 trajectories from each of those states and averaging the returns. In the absence of true values, these values were used to measure the performance of the estimators.

In order to generate data samples, 20 trajectories were generated from each of the 25 randomly chosen start states by following a behavior policy. The behavior policy was constructed by applying an ϵ-greedy policy on the learned parameters, where the value of ϵ was different than the value used with the target policy. We generated two sets of data by using $\epsilon = 0.1$ and 0.2. In each case, the return and the importance weight corresponding to each trajectory was stored.

In order to train OIS-LS and WIS-LS, we generated a set feature vectors, one for each start states, using tile coding, where the number of tiling was 10 and a feature vector of size 64. The performance was measured using NMSPE, where in the absence of a true MSE solution, we estimated it by drawing samples under the target policy. The performance was averaged using 30 different independent sets of data. Different values of the regularization parameter was tested for both estimators, the best value of this parameter was used.

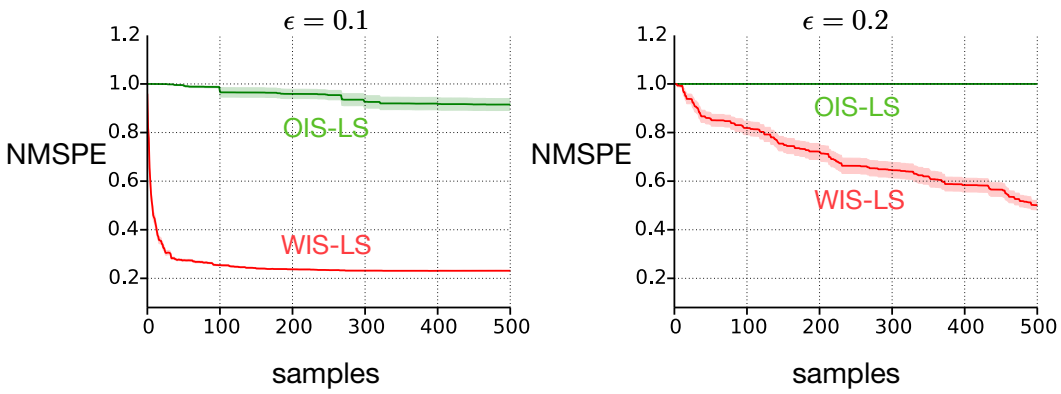

Figure 5.2: Performance comparison between OIS-LS and WIS-LS on two Mountain Car off-policy prediction tasks with two different behavior policies that are fixed ϵ-greedy policies based on parameters learned by Sarsa(0.9). Performance of both estimators are shown in Normalized Mean Squared Projected Error (NMSPE) for learning over 500 samples and averaged over 30 independently generated data sets. In both tasks WIS-LS significantly outperforms OIS-LS.

Figure 5.2 shows the performance of OIS-LS and WIS-LS for the two different behavior policies we chose. In both cases, WIS-LS substantially outperformed OIS-LS. The difference between the two plots shows that as the behavior policy deviates more from the target policy, caused by the increase in exploration in the behavior policy, learning becomes more difficult. However, even with an increase in the exploration, WIS-LS remained the only algorithm that learned effectively.

5.3 WIS2 with Linear Function Approximation

In this section, we extend WIS2, which developed in Chapter 4, to linear function approximation. The main idea here is to combine the intuition behind developing WIS2 and the intuition behind extending WIS to WIS-LS.

The intuition behind developing both OIS2 and WIS2 was to view a return as a combination of different n-step flat returns. Then the target samples for off-policy estimation are formed by scaling the individual flat returns with corresponding importance weights instead of scaling the whole return with one importance weight. This idea of scaling different flat returns can be utilized to produce the following two empirical objectives:

$$\tilde{J}_n(\hat{v}) = \frac{1}{n}\sum_{k=1}^{n}\left(\sum_{h=t_k+1}^{T_k} W_{t_k}^h\,(1-\gamma_h)\prod_{i=t_k+1}^{h-1}\gamma_i\bar{G}_{t_k}^h - \hat{v}\right)^2, \tag{5.16}$$

$$\hat{J}_n(\hat{v}) = \frac{1}{n}\sum_{k=1}^{n}\sum_{h=t_k+1}^{T_k} W_{t_k}^h\,(1-\gamma_h)\prod_{i=t_k+1}^{h-1}\gamma_i\left(\bar{G}_{t_k}^h - \hat{v}\right)^2. \tag{5.17}$$

In the first objective, the individual flat returns are scaled by the importance weights whereas in the second objective, the error for each individual flat return is scaled by the importance weight.

Then it can be easily shown that V^{WIS2} is the solution to $J_n(\hat{v})$:

$$V^{\mathrm{OIS2}} = \arg\min_v \tilde{J}_n(v) = \frac{\sum_{k=1}^n \sum_{h=t_k+1}^{T_k} W_{t_k}^h \left(1-\gamma_h\right) \prod_{i=t_k+1}^{h-1} \gamma_i \bar{G}_{t_k}^h}{n}, \tag{5.18}$$

$$V^{\mathrm{WIS2}} = \arg\min_v \hat{J}_n(v) = \frac{\sum_{k=1}^n \sum_{h=t_k+1}^{T_k} W_{t_k}^h \left(1-\gamma_h\right) \prod_{i=t_k+1}^{h-1} \gamma_i \bar{G}_{t_k}^h}{\sum_{k=1}^n \sum_{h=t_k+1}^{T_k} W_{t_k}^h \left(1-\gamma_h\right) \prod_{i=t_k+1}^{h-1} \gamma_i}. \tag{5.19}$$

An extension to the empirical objectives to linear function approximation would be as follows:

$$\tilde{J}_t(\boldsymbol{\theta}) \stackrel{\mathrm{def}}{=\!=} \frac{1}{n} \sum_{k=1}^n \left(\sum_{h=t_k+1}^{T_k} W_{t_k}^h \left(1-\gamma_h\right) \prod_{i=t_k+1}^{h-1} \gamma_i \bar{G}_{t_k}^h - \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t_k} \right)^2, \tag{5.20}$$

$$\hat{J}_t(\boldsymbol{\theta}) \stackrel{\mathrm{def}}{=\!=} \frac{1}{n} \sum_{k=1}^n \sum_{h=t_k+1}^{T_k} W_{t_k}^h \left(1-\gamma_h\right) \prod_{i=t_k+1}^{h-1} \gamma_i \left(\bar{G}_{t_k}^h - \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t_k} \right)^2. \tag{5.21}$$

We obtain the following two estimators by solving the above two empirical objectives, respectively:

$$\tilde{\boldsymbol{\theta}}_n \stackrel{\mathrm{def}}{=\!=} \left(\frac{1}{n} \sum_{k=1}^n \boldsymbol{\phi}_{t_k} \boldsymbol{\phi}_{t_k}^\top \right)^{-1} \left(\frac{1}{n} \sum_{k=1}^n \sum_{h=t_k+1}^{T_k} W_{t_k}^h \left(1-\gamma_h\right) \prod_{i=t_k+1}^{h-1} \gamma_i \bar{G}_{t_k}^h \boldsymbol{\phi}_{t_k} \right), \tag{5.22}$$

$$\hat{\boldsymbol{\theta}}_n \stackrel{\mathrm{def}}{=\!=} \left(\frac{1}{n} \sum_{k=1}^n \sum_{h=t_k+1}^{T_k} W_{t_k}^h \left(1-\gamma_h\right) \prod_{i=t_k+1}^{h-1} \gamma_i \boldsymbol{\phi}_{t_k} \boldsymbol{\phi}_{t_k}^\top \right)^{-1} \tag{5.23}$$

$$\times \left(\frac{1}{n} \sum_{k=1}^n \sum_{h=t_k+1}^{T_k} W_{t_k}^h \left(1-\gamma_h\right) \prod_{i=t_k+1}^{h-1} \gamma_i \bar{G}_{t_k}^h \boldsymbol{\phi}_{t_k} \right). \tag{5.24}$$

We call $\tilde{\boldsymbol{\theta}}$ the *OIS2-LS estimator* and $\hat{\boldsymbol{\theta}}$ the *WIS2-LS estimator*. The following theorems about OIS2-LS and WIS2-LS are similar to the theorems we have proved in Section 5.2 with regards to OIS-LS and WIS-LS. They show that both OIS2-LS and WIS2-LS are proper extensions of OIS2 and WIS2, carrying forward their essential properties to linear function approximation.

Theorem 18 (Unbiasedness of OIS2-LS)**.** *If v_π is a linear function of the features, that is, $v_\pi(s) = {\boldsymbol{\theta}_*}^\top \boldsymbol{\phi}(s)$, then OIS2-LS is an unbiased estimator, that is, $\mathrm{E}_{\mu,b_\mu}[\tilde{\boldsymbol{\theta}}_n] = \boldsymbol{\theta}_*$.*

Theorem 19 (Bias of WIS2-LS)**.** *Even if v_π is a linear function of the features, that is, $v_\pi(s) = \boldsymbol{\theta}_*^\top \boldsymbol{\phi}(s)$, WIS2-LS is in general a biased estimator, that is, $\mathrm{E}_{\mu,b_\mu}[\hat{\boldsymbol{\theta}}_n] \neq \boldsymbol{\theta}_*$.*

Theorem 20 (Consistency of OIS2-LS)**.** *The OIS2-LS estimator $\tilde{\boldsymbol{\theta}}_n$ is a consistent estimator of the MSE solution $\boldsymbol{\theta}_*$ given in (5.11).*

Theorem 21 (Consistency of WIS2-LS)**.** *The WIS2-LS estimator $\hat{\boldsymbol{\theta}}_n$ is a consistent estimator of the MSE solution $\boldsymbol{\theta}_*$ given in (5.11).*

Theorem 22 (Backward compatibility of OIS2-LS with OIS2)**.** *If the features form an orthonormal basis, then the OIS2-LS estimate $\tilde{\boldsymbol{\theta}}_n^\top \boldsymbol{\phi}(s)$ is equivalent to the OIS2 estimate of $v_\pi(s)$.*

Theorem 23 (Backward compatibility of WIS2-LS with WIS2)**.** *If the features form an orthonormal basis, then the WIS2-LS estimate $\hat{\boldsymbol{\theta}}_n^\top \boldsymbol{\phi}(s)$ of state s is equivalent to the WIS2 estimate of the value of s.*

Proofs of these theorems are similar to those in Section 5.2 and we leave them out here.

5.4 Conclusions

In this chapter, we developed some key intuitions for extending tabular estimators to linear function approximation in a backward compatible way. This allowed us to provide the first weighted importance sampling method for linear function approximation. We showed through theoretical and empirical results that our new methods carry over the beneficial properties of weighted importance sampling to linear function approximation. Finally, we performed a similar extension to a superior version of weighted importance sampling, WIS2, resulting in a new algorithm we called WIS2-LS. This method builds the foundation for more powerful and computationally amenable off-policy algorithms.

Chapter 6

Real-Time Weighted Importance Sampling with Bootstrapping [1]

In this chapter, we provide two main contributions of this thesis. First, we provide a novel and systematic technique for deriving algorithms based on algorithmic equivalence. This builds on some of the intuitions illustrated in Chapter 3. Second, we provide the fullest extension of the new weighted importance sampling estimator—WIS2—we developed in Chapter 4. This results into a new strictly incremental off-policy algorithm for the real-time learning setting, we call WIS-LSTD(λ). This is the culmination of the series of contributions we make in this thesis based on weighted importance sampling toward the issue of high variance.

In Chapter 5, we extended WIS2 to the case of linear function approximation to develop a least-squares method based on per-trajectory updates. Our new algorithm surpasses the contributions of the previous chapter in two other important ways. First, the new algorithm goes beyond the Monte Carlo updates of the algorithms from the previous chapter by incorporating *bootstrapping*, a well-known technique for reducing variance in reinforcement learning. Second, the algorithms in the previous chapter were based on per-trajectory updates, whereas the new algorithm can be updated strictly incrementally on a real-time basis. The question of how a per-trajectory algorithm can be extended to a real-time algorithm is an intricate topic, and Chapter 3 only scratches the surface of it. In this chapter, we take a closer look at it and devise a principled way of performing this feat.

6.1 Forming Targets with State-dependent Bootstrapping

Bootstrapping is a sophisticated technique used in many reinforcement-learning learning algorithms where the estimates are formed using the estimates of the subsequent states rather than solely using samples as in Monte Carlo estimates. One of the most popular

[1] The contribution toward algorithmic equivalence techniques is adapted from the techniques developed in a published paper coauthored by this author (Mahmood & Sutton 2015). The contribution toward the fullest extension of weighted importance sampling is adapted from a published paper coauthored by this author (Mahmood, van Hasselt & Sutton 2014).

reinforcement-learning algorithm–the Temporal-Difference (TD) learning algorithm—is the quintessential example of bootstrapping. By avoiding updates solely based on direct sampling, bootstrapping estimates can reduce variance in exchange for introduced bias in the finite sample case. Bootstrapping estimates can be asymptotically unbiased in the case of lookup-table representation, however, in the case of parametric function approximation, they are generally biased.

As Monte Carlo estimates are formed using returns as targets, bootstrapping estimates are formed using modified returns where a part of it is formed using samples and the rest is formed using the estimate of the subsequent states. The simplest of such targets use only the immediate reward as the sample and the estimate of the next state, which we call *the one-step flat return*: $R_{t+1} + \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t+1}$. The estimate of the next state uses weight estimate $\boldsymbol{\theta}$. Bootstrapping estimates make a specific choice for the weight estimates. In this section and the next, we will only form targets without committing to a specific bootstrapping estimate.

In general, an *n-step corrected flat return* can be defined in the following way:

$$\bar{G}_t^{(n)} \stackrel{\text{def}}{=\!=} \bar{G}_t^{t+n} + \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t+n} = \sum_{k=t+1}^{t+n} R_k + \boldsymbol{\theta}^\top \phi_{t+n}, \tag{6.1}$$

where $\bar{G}_t^h$ is the flat return defined by (4.27). The n-step corrected flat return incorporates a *bootstrapped estimate* $\boldsymbol{\theta}^\top \boldsymbol{\phi}_{t+n}$ with an incomplete flat return $\bar{G}_t^{t+n}$ at the horizon $t+n$. However, it is not discounting aware. We can include discounting awareness using the technique given in Section 4.2.

A more powerful target for updates would be one that combines different bootstrapping targets as well as the full Monte Carlo return in a single target. This can be achieved by exponentially scaling different n-step corrected returns using a parameter most commonly denoted by $\lambda \in [0, 1]$, often known as *the bootstrapping parameter*. Such combined targets are typically formed when both the discount factor and the bootstrapping parameter are constants. As the discount factor can be state dependent and viewed as the degree of termination, a similar interpretation can be applied to λ by allowing them to depend on states and view them as the degree of bootstrapping that can vary with states.

In the following, we introduce a combined return, where different n-step corrected flat return are weighted according to their corresponding degree of discounting, bootstrapping, as well as importance weights. We start by the first uncorrected flat return $\bar{G}_t^{t+1} = R_{t+1}$, which stands for a trajectory up to horizon $t+1$. Of course, the trajectory may not be complete in a single transition. But this incompleteness of the return can be accounted for by scaling it with the degree of termination corresponding to this trajectory given by $1 - \gamma_{t+1}$. Therefore, a valid off-policy target corresponding to this return can be obtained by scaling it with the degree of termination at time $t+1$ times the importance weight corresponding to this trajectory: $(1 - \gamma_{t+1}) W_t^{t+1}$. On the other hand, the first corrected flat return $\bar{G}_t^{t+1} + \boldsymbol{\theta}^\top \boldsymbol{\phi}_{t+1}$ stands for a trajectory which did not terminate at time $t+1$, but

has rather used bootstrapping. The degree of continuation at time $t+1$ is given by γ_{t+1}, and the degree of bootstrapping at $t+1$ is given by $1-\lambda_{t+1}$. Therefore, the corresponding weights for this return would be the degree of continuation at time $t+1$ times the degree of bootstrapping at time $t+1$ times the importance weight: $\gamma_{t+1}(1-\lambda_{t+1})W_t^{t+1}$. By using this idea for the subsequent returns, the combined return, which we call the λ-return, can be formed as follows:

$$G_t^\lambda(\boldsymbol{\theta}) = (1-\gamma_{t+1})W_t^{t+1}\bar{G}_t^{t+1} + \gamma_{t+1}(1-\lambda_{t+1})W_t^{t+1}\left(\bar{G}_t^{t+1} + \boldsymbol{\theta}^\top\boldsymbol{\phi}_{t+1}\right) \tag{6.2}$$

$$+ \gamma_{t+1}\lambda_{t+1}(1-\gamma_{t+2})W_t^{t+2}\bar{G}_t^{t+2} \tag{6.3}$$

$$+ \gamma_{t+1}\lambda_{t+1}\gamma_{t+2}(1-\lambda_{t+2})W_t^{t+2}\left(\bar{G}_t^{t+2} + \boldsymbol{\theta}^\top\boldsymbol{\phi}_{t+2}\right) + \cdots \tag{6.4}$$

$$= \sum_{h=t+1}^{\infty} \gamma_{t+1}^{h-1}\lambda_{t+1}^{h-1}W_t^h\left[(1-\gamma_h)\bar{G}_t^h + (1-\lambda_h)\gamma_h\left(\bar{G}_t^h + \boldsymbol{\theta}^\top\boldsymbol{\phi}_h\right)\right]. \tag{6.5}$$

Note that, at $T(t)$, the trajectory must fully terminate: $\gamma_{T(t)} = 0$. Therefore, the above return can be rewritten as:

$$G_t^\lambda = \sum_{h=t+1}^{T(t)-1} \gamma_{t+1}^{h-1}\lambda_{t+1}^{h-1}W_t^h\left[(1-\gamma_h)\bar{G}_t^h + (1-\lambda_h)\gamma_h\left(\bar{G}_t^h + \boldsymbol{\theta}^\top\boldsymbol{\phi}_h\right)\right] \tag{6.6}$$

$$+ \gamma_{t+1}^{T(t)-1}\lambda_{t+1}^{T(t)-1}W_t^{T(t)}\bar{G}_t^{T(t)}. \tag{6.7}$$

Note that we have scaled the returns by the importance weights directly instead of scaling errors. The question of whether to scale the returns or the errors is still relevant here, which we will consider in a later section. Scaling the return is the simplest and the most ordinary way of applying importance sampling, and we have adopted it here to introduce the complex return in a simple way.

6.2 Interim Targets for Real-time Updates

In this section, we extend the complex return G_t^λ so that updates can be made real time toward a meaning target without waiting for a trajectory to terminate fully. So far, all the off-policy updates we have discussed have been on a per-trajectory basis. The complex return we formed in the previous section also completes upon full termination at $T(t)$. However, to make real-time updates, we need to form a target that is valid and meaningful at any time. We expand here the concept of *interim* targets introduced in Section 3.2. In that section, the interim targets did not consider complex returns and avoided the intricacy of bootstrapping on the estimate of subsequent states by considering bootstrapping on open-ended "guesses". We address those issues here.

Let us consider that the current time step is $t < T(t)$. Ideally, an interim target would use the samples from time step k up to interim horizon t but would be incomplete otherwise. If we adopt the λ-return defined by (8.6) for this, it is clear that the summation would need

to be truncated at the interim horizon t. All the prior summands can be kept in their original form, but we have to determine what should be the last summand. As the return needs to appear complete to the update, we can choose to fully bootstrap at the interim horizon. Hence, the *interim return* for an interim horizon t can be defined as:

$$G^{\lambda}_{k,t} = \sum_{h=k+1}^{t-1} \gamma^{h-1}_{k+1}\lambda^{h-1}_{k+1}W^{h}_{k}\left[(1-\gamma_h)\bar{G}^{h}_{k} + (1-\lambda_h)\gamma_h\left(\bar{G}^{h}_{k} + \boldsymbol{\theta}^{\top}\boldsymbol{\phi}_h\right)\right] \tag{6.8}$$

$$+ \gamma^{t-1}_{k+1}\lambda^{t-1}_{k+1}W^{t}_{k}\left[(1-\gamma_t)\bar{G}^{t}_{k} + \gamma_t\left(\bar{G}^{t}_{k} + \boldsymbol{\theta}^{\top}\boldsymbol{\phi}_t\right)\right]. \tag{6.9}$$

Note that this interim target becomes equivalent to G^{λ}_{k} at full termination. The interim return $G^{\lambda}_{k,t}$ provides a valid target for making updates at all time t for all past visited states at time $k < t$ without waiting for the trajectory to complete.

6.3 Putting It All Together

In this section, we form an off-policy algorithm with real-time updates based on interim targets. The resulting algorithm here is of least-squares form. The novel problem we face here is developing a learning algorithm that makes updates for a parameter vector $\boldsymbol{\theta}$ toward targets with bootstrapped estimates, which themselves are based on the same parameter vector $\boldsymbol{\theta}$. There are two notable approaches that can be taken in this case. We can choose the parameter vector for the bootstrapped estimates to be exactly the same as the current estimate or set it at the solution of the least-squares update. The former approach is known as the *residual approach* whereas the latter is known as the *projected fixed point approach* (Geist and Scherrer 2014). We take the latter approach here.

As we utilize the interim targets to achieve real-time updates, we abandon the per-trajectory based time-step notation t_k. Instead, we consider all the interim returns from each of the intermediate time steps starting from time 0 up to time t. As in previous, we face two alternative routes to forming an off-policy least-squares algorithm: scale the returns directly or scale the errors instead, where the former leads to an ordinary form of importance sampling estimator whereas the latter leads to a weighted importance sampling estimator. As we have already found that weighted importance sampling estimators are preferable to its ordinary counterpart, we take only the latter approach here.

By utilizing the idea behind forming the interim return defined by (6.9), we form an empirical objective function where the errors are scaled by the corresponding degree of discounting, bootstrapping, and importance weights. Such an objective function can be defined as:

$$J_t(\boldsymbol{\theta}, \mathbf{v}) \stackrel{\text{def}}{=\!=} \frac{1}{t}\sum_{k=0}^{t-1}\ell_{k,t}(\boldsymbol{\theta}, \mathbf{v}), \tag{6.10}$$

$$\ell_{k,t}(\boldsymbol{\theta}, \mathbf{v}) \stackrel{\text{def}}{=\!=} \sum_{h=k+1}^{t-1} \gamma^{h-1}_{k+1}\lambda^{h-1}_{k+1}W^{h}_{k}\Bigg[(1-\gamma_h)\left(\bar{G}^{h}_{k} - \boldsymbol{\theta}^{\top}\boldsymbol{\phi}_k\right)^2 \tag{6.11}$$

$$+ (1-\lambda_h)\gamma_h \left(\bar{G}_k^h + \mathbf{v}^\top \boldsymbol{\phi}_h - \boldsymbol{\theta}^\top \boldsymbol{\phi}_k\right)^2 \Bigg] \tag{6.12}$$

$$+ \gamma_{k+1}^{t-1}\lambda_{k+1}^{t-1} W_k^t \left[(1-\gamma_t)\left(\bar{G}_k^t - \mathbf{v}^\top \boldsymbol{\phi}_k\right)^2 + \gamma_t \left(\bar{G}_k^t + \mathbf{v}^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}^\top \boldsymbol{\phi}_k\right)^2\right]. \tag{6.13}$$

By choosing the bootstrapped estimate to have values at the solution, the solution is defined as follows:

$$\boldsymbol{\theta}_t = \arg\min_{\boldsymbol{\theta}} J_t(\boldsymbol{\theta}, \boldsymbol{\theta}_t). \tag{6.14}$$

In order to find the solution in a closed form, we write out the expression for which the minimum of the objective is achieved, and separate all the terms that involve $\boldsymbol{\theta}_t$ from those that do not. At the solution, the gradient of the objective equates to zero:

$$\nabla_{\boldsymbol{\theta}} J_t(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} = \frac{1}{t}\sum_{k=0}^{t-1} \ell_{k,t}(\boldsymbol{\theta}, \mathbf{v}) = -2\frac{1}{t}\sum_{k=0}^{t-1} \delta_{k,t}(\boldsymbol{\theta}_t, \boldsymbol{\theta}_t)\boldsymbol{\phi}_k = \mathbf{0}, \tag{6.15}$$

where the errors $\delta_{k,t}$ are defined by

$$\delta_{k,t}(\boldsymbol{\theta}, \mathbf{v}) = \sum_{h=k+1}^{t-1} \gamma_{k+1}^{h-1}\lambda_{k+1}^{h-1} W_k^h \left[(1-\gamma_h)\left(\bar{G}_k^h - \boldsymbol{\theta}^\top \boldsymbol{\phi}_k\right) + (1-\lambda_h)\gamma_h \left(\bar{G}_k^h + \mathbf{v}^\top \boldsymbol{\phi}_h - \boldsymbol{\theta}^\top \boldsymbol{\phi}_k\right)\right] \tag{6.16}$$

$$+ \gamma_{k+1}^{t-1}\lambda_{k+1}^{t-1} W_k^t \left[(1-\gamma_t)\left(\bar{G}_k^t - \mathbf{v}^\top \boldsymbol{\phi}_k\right) + \gamma_t \left(\bar{G}_k^t + \mathbf{v}^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}^\top \boldsymbol{\phi}_k\right)\right]. \tag{6.17}$$

By separating the terms of $\delta_{k,t}(\boldsymbol{\theta}_t, \boldsymbol{\theta}_t)\boldsymbol{\phi}_k$ with $\boldsymbol{\theta}_t$ from those without, we obtain:

$$\delta_{k,t}(\boldsymbol{\theta}_t, \boldsymbol{\theta}_t)\boldsymbol{\phi}_k = \tilde{\mathbf{b}}_{k,t} - \tilde{\mathbf{A}}_{k,t}\boldsymbol{\theta}_t. \tag{6.18}$$

Here, $\tilde{\mathbf{b}}_{k,t} \in \mathbb{R}^m, \tilde{\mathbf{A}}_{k,t} \in \mathbb{R}^{m\times m}$, and they are defined as

$$\tilde{\mathbf{b}}_{k,t} = \sum_{h=k+1}^{t-1} \gamma_{k+1}^{h-1}\lambda_{k+1}^{h-1} W_k^h \left[(1-\gamma_h)\bar{G}_k^h + (1-\lambda_h)\gamma_h \bar{G}_k^h\right]\boldsymbol{\phi}_k \tag{6.19}$$

$$+ \gamma_{k+1}^{t-1}\lambda_{k+1}^{t-1} W_k^t \left[(1-\gamma_t)\bar{G}_k^t + \gamma_t \bar{G}_k^t\right]\boldsymbol{\phi}_k \tag{6.20}$$

$$= \sum_{h=k+1}^{t-1} \gamma_{k+1}^{h-1}\lambda_{k+1}^{h-1} W_k^h (1-\gamma_h\lambda_h)\bar{G}_k^h \boldsymbol{\phi}_k + \gamma_{k+1}^{t-1}\lambda_{k+1}^{t-1} W_k^t \bar{G}_k^t \boldsymbol{\phi}_k, \tag{6.21}$$

$$\tilde{\mathbf{A}}_{k,t} = \sum_{h=k+1}^{t-1} \gamma_{k+1}^{h-1}\lambda_{k+1}^{h-1} W_k^h \boldsymbol{\phi}_k \left[(1-\gamma_h)\boldsymbol{\phi}_k^\top - (1-\lambda_h)\gamma_h \left(\boldsymbol{\phi}_h - \boldsymbol{\phi}_k\right)^\top\right] \tag{6.22}$$

$$+ \gamma_{k+1}^{t-1}\lambda_{k+1}^{t-1} W_k^t \boldsymbol{\phi}_k \left[(1-\gamma_t)\boldsymbol{\phi}_k^\top - \gamma_t \left(\boldsymbol{\phi}_t - \boldsymbol{\phi}_k\right)^\top\right] \tag{6.23}$$

$$= \sum_{h=k+1}^{t-1} \gamma_{k+1}^{h-1}\lambda_{k+1}^{h-1} W_k^h \boldsymbol{\phi}_k \left[(1-\gamma_h\lambda_h)\boldsymbol{\phi}_k - \gamma_h(1-\lambda_h)\boldsymbol{\phi}_h\right]^\top + \gamma_{k+1}^{t-1}\lambda_{k+1}^{t-1} W_k^t \boldsymbol{\phi}_k \left[\boldsymbol{\phi}_k - \gamma_t\boldsymbol{\phi}_t\right]^\top. \tag{6.24}$$

Therefore, the solution can be written in a closed form as follows:

$$\frac{1}{t}\sum_{k=0}^{t-1}\left(\tilde{\mathbf{b}}_{k,t}-\tilde{\mathbf{A}}_{k,t}\boldsymbol{\theta}_t\right)=\mathbf{0}\implies\boldsymbol{\theta}_t=\mathbf{A}_t^{-1}\mathbf{b}_t, \tag{6.25}$$

where

$$\mathbf{A}_t=\frac{1}{t}\sum_{k=0}^{t-1}\tilde{\mathbf{A}}_{k,t}, \tag{6.26}$$

$$\mathbf{b}_t=\frac{1}{t}\sum_{k=0}^{t-1}\tilde{\mathbf{b}}_{k,t}. \tag{6.27}$$

We call this algorithm the *WIS-LSTD*(λ) algorithm, because this is a generalization of LSTD(λ) algorithm (Boyan 2002) with WIS. When $\lambda_k=1,\forall k$ and termination occurs then the solution reduces to that of WIS2-LS, which we show in the following.

Theorem 24 (Backward compatibility of WIS-LSTD(λ) with WIS2-LS)**.** *At termination ($\gamma_t=0$) with $\lambda_k=1,\forall k$, and $t_k=k$, the solution defined by (6.25) is equivalent to the WIS2-LS solution defined by (5.24).*

Proof. By choosing $t_k=k$, $\gamma_t=0$, and $\lambda_k=1,\forall k$, we can rewrite $\mathbf{b}_t$ and $\mathbf{A}_t$ as

$$\mathbf{b}_t=\frac{1}{t}\sum_{k=0}^{t-1}\left(\sum_{h=k+1}^{t-1}\gamma_{k+1}^{h-1}W_k^h(1-\gamma_h)\bar{G}_k^h\boldsymbol{\phi}_k+\gamma_{k+1}^{t-1}W_k^t\bar{G}_k^t\boldsymbol{\phi}_k\right) \tag{6.28}$$

$$=\frac{1}{t}\sum_{k=0}^{t-1}\sum_{h=k+1}^{t}\gamma_{k+1}^{h-1}W_k^h(1-\gamma_h)\bar{G}_k^h\boldsymbol{\phi}_k, \tag{6.29}$$

$$\mathbf{A}_t=\frac{1}{t}\sum_{k=0}^{t-1}\sum_{h=k+1}^{t-1}\gamma_{k+1}^{h-1}W_k^h(1-\gamma_h)\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top+\gamma_{k+1}^{t-1}W_k^t\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top \tag{6.30}$$

$$=\frac{1}{t}\sum_{k=0}^{t-1}\sum_{h=k+1}^{t}\gamma_{k+1}^{h-1}W_k^h(1-\gamma_h)\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top. \tag{6.31}$$

Also note that $T_k=T(t_k)=T(k)$, which we can also use to replace the upper limit of the inner summation for both $\mathbf{b}_t$ and $\mathbf{A}_t$, because the time-step t is either the first time step after k where a termination occurred: $T(k)=t$ or there are other terminations before t: $T(k)<t$. Then it is clear that the solution $\boldsymbol{\theta}_t=\mathbf{A}_t^{-1}\mathbf{b}_t$ coincides with the solution of WIS2-lS. □

6.4 Strictly Incremental Updates with Algorithmic Equivalence Technique

In this section, we derive a strictly incremental update that produces the same solutions as the real-time update in the previous section. The crux of the derivation is switching

the order of the double summations as in Lemma 8 and seeking recursive expressions. As we use these derivation steps multiple times for different algorithms in this work, it would be desirable to avoid much of the repetitions by providing a general technique, which we can instantiate in each case. This general technique would capture the common specialties involved in the reinforcement learning updates toward complex interim targets such as λ-returns.

A key feature in complex interim returns, on which all strict incremental updates rely on, is the recursive expressions of the returns in the expanding horizon. A return with interim horizon $t+1$ can typically be written recursively in terms of the return with interim horizon t. The following return is a general form for all the complex interim targets we have used in this work:

$$Y_k^{t+1} - Y_k^t = d_{k+1}\left(Y_{k+1}^{t+1} - Y_{k+1}^t\right) + b_t g_k \prod_{j=k+1}^{t-1} c_j, 0 \leq k \leq t, \tag{6.32}$$

where μ_k, c_k, d_k, and g_k are scalars that can be computed using data available at time k. We show in the following that an update toward the target Y_k^{t+1} can then be computed strictly incrementally.

Theorem 25 (Algorithmic equivalence theorem)**.** *Consider any forward view that updates toward an interim target Y_k^t with*

$$\boldsymbol{\theta}_{k+1}^{t+1} \stackrel{\text{def}}{=\!=} \mathbf{F}_k \boldsymbol{\theta}_k^{t+1} + Y_k^{t+1}\mathbf{w}_k + \mathbf{x}_k, \quad 0 \leq k < t+1,$$

where $\boldsymbol{\theta}_0^t \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_0$ for some initial $\boldsymbol{\theta}_0$, and both $\mathbf{F}_k \in \mathbb{R}^{m\times m}$ and $\mathbf{w}_k \in \mathbb{R}^m$ can be computed using data available at k. Assume that the temporal difference $Y_k^{t+1} - Y_k^t$ at k is related to the temporal difference at $k+1$ as follows:

$$Y_k^{t+1} - Y_k^t = d_{k+1}\left(Y_{k+1}^{t+1} - Y_{k+1}^t\right) + b_t g_k \prod_{j=k+1}^{t-1} c_j, 0 \leq k < t,$$

where b_k, c_k, d_k and g_k can be computed using data available at time k. Then the final weight $\boldsymbol{\theta}_{t+1} \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_{t+1}^{t+1}$ can be computed through the following backward-view updates, with $\mathbf{e}_{-1} \stackrel{\text{def}}{=\!=} \mathbf{0}$, $\mathbf{d}_0 \stackrel{\text{def}}{=\!=} \mathbf{0}$, and $t \geq 0$:

$$\begin{aligned}
\mathbf{e}_t &\stackrel{\text{def}}{=\!=} \mathbf{w}_t + d_t \mathbf{F}_t \mathbf{e}_{t-1}, \\
\boldsymbol{\theta}_{t+1} &\stackrel{\text{def}}{=\!=} \mathbf{F}_t \boldsymbol{\theta}_t + (Y_t^{t+1} - Y_t^t)\mathbf{e}_t + Y_t^t \mathbf{w}_t + b_t \mathbf{F}_t \mathbf{d}_t + \mathbf{x}_t, \\
\mathbf{d}_{t+1} &\stackrel{\text{def}}{=\!=} c_t \mathbf{F}_t \mathbf{d}_t + g_t \mathbf{e}_t.
\end{aligned}$$

Proof. We can write the difference between two consecutive estimates as

$$\begin{aligned}
\boldsymbol{\theta}_{t+1}^{t+1} - \boldsymbol{\theta}_t^t &= \mathbf{F}_t \boldsymbol{\theta}_t^{t+1} - \boldsymbol{\theta}_t^t + Y_t^{t+1}\mathbf{w}_k + \mathbf{x}_t \\
&= \mathbf{F}_t\left(\boldsymbol{\theta}_t^{t+1} - \boldsymbol{\theta}_t^t\right) + Y_t^{t+1}\mathbf{w}_k + (\mathbf{F}_t - \mathbf{I})\boldsymbol{\theta}_t^t + \mathbf{x}_t.
\end{aligned}$$

Now let us expand $\boldsymbol{\theta}_t^{t+1} - \boldsymbol{\theta}_t^t$:

$$
\begin{aligned}
\boldsymbol{\theta}_t^{t+1} - \boldsymbol{\theta}_t^t &= \mathbf{F}_{t-1}\boldsymbol{\theta}_{t-1}^{t+1} + Y_{t-1}^{t+1}\mathbf{w}_{t-1} + \mathbf{x}_{t-1} \\
&\quad - \mathbf{F}_{t-1}\boldsymbol{\theta}_{t-1}^{t} - Y_{t-1}^{t}\mathbf{w}_{t-1} - \mathbf{x}_{t-1} \\
&= \mathbf{F}_{t-1}\left(\boldsymbol{\theta}_{t-1}^{t+1} - \boldsymbol{\theta}_{t-1}^{t}\right) + \left(Y_{t-1}^{t+1} - Y_{t-1}^{t}\right)\mathbf{w}_{t-1} \\
&= \mathbf{F}_{t-1}\cdots\mathbf{F}_0(\boldsymbol{\theta}_0^{t+1} - \boldsymbol{\theta}_0^t) + \sum_{k=0}^{t-1}\mathbf{F}_{t-1}\cdots\mathbf{F}_{k+1}(Y_k^{t+1} - Y_k^t)\mathbf{w}_k \\
&= \sum_{k=0}^{t-1}\mathbf{F}_{t-1}\cdots\mathbf{F}_{k+1}(Y_k^{t+1} - Y_k^t)\mathbf{w}_k \\
&= \sum_{k=0}^{t-1}\mathbf{F}_{t-1}\cdots\mathbf{F}_{k+1}\left(d_{k+1}(Y_{k+1}^{t+1} - Y_{k+1}^t) + b_t g_k \prod_{j=k+1}^{t-1} c_j\right)\mathbf{w}_k \\
&= \sum_{k=0}^{t-1}\mathbf{F}_{t-1}\cdots\mathbf{F}_{k+1}\Bigg(d_{k+1}\Bigg(d_{k+2}(Y_{k+2}^{t+1} - Y_{k+2}^t) \\
&\qquad + b_t g_{k+1}\prod_{j=k+2}^{t-1} c_j\Bigg) + b_t g_k \prod_{j=k+1}^{t-1} c_j\Bigg)\mathbf{w}_k \\
&= \sum_{k=0}^{t-1}\mathbf{F}_{t-1}\cdots\mathbf{F}_{k+1}\Bigg(d_{k+1}d_{k+2}(Y_{k+2}^{t+1} - Y_{k+2}^t) \\
&\qquad + b_t g_{k+1}d_{k+1}\prod_{j=k+2}^{t-1} c_j + b_t g_k \prod_{j=k+1}^{t-1} c_j\Bigg)\mathbf{w}_k \\
&= \sum_{k=0}^{t-1}\mathbf{F}_{t-1}\cdots\mathbf{F}_{k+1}\Bigg(\prod_{j=k+1}^{t} d_j(Y_t^{t+1} - Y_t^t) \\
&\qquad + b_t\sum_{n=k}^{t-1} g_n \prod_{i=k+1}^{n} d_i \prod_{j=n+1}^{t-1} c_j\Bigg)\mathbf{w}_k \\
&= d_t(Y_t^{t+1} - Y_t^t)\underbrace{\sum_{k=0}^{t-1}\mathbf{F}_{t-1}\cdots\mathbf{F}_{k+1}\prod_{j=k+1}^{t-1} d_j\mathbf{w}_k}_{\mathbf{e}_{t-1}} \\
&\quad + b_t\underbrace{\sum_{k=0}^{t-1}\mathbf{F}_{t-1}\cdots\mathbf{F}_{k+1}\sum_{n=k}^{t-1} g_n \prod_{i=k+1}^{n} d_i \prod_{j=n+1}^{t-1} c_j\mathbf{w}_k}_{\mathbf{d}_t} \\
&= (Y_t^{t+1} - Y_t^t)d_t\mathbf{e}_{t-1} + b_t\mathbf{d}_t.
\end{aligned}
$$

The vectors $\mathbf{e}_t$ and $\mathbf{d}_t$ can be incrementally updated as follows:

$$
\mathbf{e}_t = \sum_{k=0}^{t}\mathbf{F}_t\cdots\mathbf{F}_{k+1}\prod_{j=k+1}^{t} d_j\mathbf{w}_k
$$

$$
\begin{aligned}
&= \mathbf{w}_t + d_t \mathbf{F}_t \sum_{k=0}^{t-1} \mathbf{F}_{t-1} \cdots \mathbf{F}_{k+1} \prod_{j=k+1}^{t-1} d_j \mathbf{w}_k \\
&= \mathbf{w}_t + d_t \mathbf{F}_t \mathbf{e}_{t-1},
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{d}_t &= \sum_{k=0}^{t-1} \mathbf{F}_{t-1} \cdots \mathbf{F}_{k+1} \sum_{n=k}^{t-1} g_n \prod_{i=k+1}^{n} d_i \prod_{j=n+1}^{t-1} c_j \mathbf{w}_k \\
&= \sum_{k=0}^{t-1} \mathbf{F}_{t-1} \cdots \mathbf{F}_{k+1} \left(\sum_{n=k}^{t-2} g_n \prod_{i=k+1}^{n} d_i \prod_{j=n+1}^{t-1} c_j \mathbf{w}_k + g_{t-1} \prod_{j=k+1}^{t-1} d_j \mathbf{w}_k \right) \\
&= \sum_{k=0}^{t-1} \mathbf{F}_{t-1} \cdots \mathbf{F}_{k+1} \sum_{n=k}^{t-2} g_n \prod_{i=k+1}^{n} d_i \prod_{j=n+1}^{t-1} c_j \mathbf{w}_k + g_{t-1} \sum_{k=0}^{t-1} \mathbf{F}_{t-1} \cdots \mathbf{F}_{k+1} \prod_{j=k+1}^{t-1} d_j \mathbf{w}_k \\
&= c_{t-1} \mathbf{F}_{t-1} \sum_{k=0}^{t-2} \mathbf{F}_{t-1} \cdots \mathbf{F}_{k+1} \sum_{n=k}^{t-2} g_n \prod_{i=k+1}^{n} d_i \prod_{j=n+1}^{t-2} c_j \mathbf{w}_k + g_{t-1} \mathbf{e}_{t-1} \\
&= c_{t-1} \mathbf{F}_{t-1} \mathbf{d}_{t-1} + g_{t-1} \mathbf{e}_{t-1}.
\end{aligned}
$$

Then plugging back in

$$
\begin{aligned}
\boldsymbol{\theta}_{t+1}^{t+1} &= \boldsymbol{\theta}_t^t + \mathbf{F}_t \left(\boldsymbol{\theta}_t^{t+1} - \boldsymbol{\theta}_t^t \right) + Y_t^{t+1} \mathbf{w}_t + (\mathbf{F}_t - \mathbf{I}) \boldsymbol{\theta}_t^t + \mathbf{x}_t \\
&= \boldsymbol{\theta}_t^t + d_t \mathbf{F}_t \mathbf{e}_{t-1} (Y_t^{t+1} - Y_t^t) + b_t \mathbf{F}_t \mathbf{d}_t + Y_t^{t+1} \mathbf{w}_t + (\mathbf{F}_t - \mathbf{I}) \boldsymbol{\theta}_t^t + \mathbf{x}_t \\
&= \mathbf{F}_t \boldsymbol{\theta}_t^t + (\mathbf{e}_t - \mathbf{w}_t)(Y_t^{t+1} - Y_t^t) + Y_t^{t+1} \mathbf{w}_t + b_t \mathbf{F}_t \mathbf{d}_t + \mathbf{x}_t \\
&= \mathbf{F}_t \boldsymbol{\theta}_t^t + (Y_t^{t+1} - Y_t^t) \mathbf{e}_t + Y_t^t \mathbf{w}_t + b_t \mathbf{F}_t \mathbf{d}_t + \mathbf{x}_t.
\end{aligned}
$$

□

This result also applies to matrix $\boldsymbol{\theta}$, vector Y and vector g, which we use in the next to derive strictly incremental updates of $\mathbf{A}_t$ and $\mathbf{b}_t$. This leads to the following strictly incremental update of WIS-LSTD(λ):

$$
\mathbf{A}_{t+1} = \left(1 - \frac{1}{t+1}\right) \mathbf{A}_t + \frac{1}{t+1} \mathbf{e}_t \left(\boldsymbol{\phi}_t - \gamma_{t+1} \boldsymbol{\phi}_{t+1}\right)^\top + \left(W_t^{t+1} - 1\right) \frac{1}{t+1} \mathbf{V}_t, \tag{6.33}
$$

$$
\mathbf{e}_t = W_t^{t+1} \boldsymbol{\phi}_t + \gamma_t \lambda_t W_t^{t+1} \mathbf{e}_{t-1}, \tag{6.34}
$$

$$
\mathbf{V}_{t+1} = \gamma_{t+1} \lambda_{t+1} \left(W_t^{t+1} \mathbf{V}_t + \mathbf{e}_t \left(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{t+1}\right)^\top \right), \tag{6.35}
$$

$$
\mathbf{b}_{t+1} = \left(1 - \frac{1}{t+1}\right) \mathbf{b}_t + \frac{1}{t+1} R_{t+1} \mathbf{e}_t + \left(W_t^{t+1} - 1\right) \frac{1}{t+1} \mathbf{u}_t, \tag{6.36}
$$

$$
\mathbf{u}_{t+1} = \gamma_{t+1} \lambda_{t+1} \left(W_t^{t+1} \mathbf{u}_t + R_{t+1} \mathbf{e}_t \right), \tag{6.37}
$$

$$
\boldsymbol{\theta}_{t+1} = \mathbf{A}_{t+1}^{-1} \mathbf{b}_{t+1}. \tag{6.38}
$$

In the following, we show that the above update is equivalent to the original real-time update of WIS-LSTD(λ).

Theorem 26 (Equivalence of strictly incremental update of WIS-LSTD(λ))**.** *The strictly incremental updates of WIS-LSTD(λ) computed by (6.33)-(6.38) are equivalent to updates computed by (6.21), (6.24), (6.25), (6.26), and (6.27).*

Proof. It suffices to show that the strictly incremental updates of $\mathbf{A}_t$ and $\mathbf{b}_t$ are equivalent to their original real-time updates. Let us define $\mathbf{A}_k^t$, which truncates the summation in $\mathbf{A}_t$ up to time k using horizon up to time t as

$$\mathbf{A}_{k+1}^{t+1} = \frac{1}{k+1}\sum_{n=0}^{k}\tilde{\mathbf{A}}_{n,t+1} = \left(1-\frac{1}{k+1}\right)\frac{1}{k}\sum_{n=0}^{k-1}\tilde{\mathbf{A}}_{n,t+1} + \frac{1}{k+1}\tilde{\mathbf{A}}_{k,t+1} \tag{6.39}$$

$$= \left(1-\frac{1}{k+1}\right)\mathbf{A}_k^{t+1} + \frac{1}{k+1}W_k^{k+1}\boldsymbol{\phi}_k\mathbf{g}_{k,t+1}^\top, \tag{6.40}$$

where $\tilde{\mathbf{A}}_{k,t} = \frac{1}{k}W_k^{k+1}\boldsymbol{\phi}_k\mathbf{g}_{k,t+1}^\top$, $\mathbf{g}_{t,t} = \mathbf{0}$, and

$$\mathbf{g}_{k,t} = \sum_{h=k+1}^{t-1}\gamma_{k+1}^{h-1}\lambda_{k+1}^{h-1}W_{k+1}^h\Big[(1-\gamma_h\lambda_h)\boldsymbol{\phi}_k - \gamma_h(1-\lambda_h)\boldsymbol{\phi}_h\Big] + \gamma_{k+1}^{t-1}\lambda_{k+1}^{t-1}W_{k+1}^t\left[\boldsymbol{\phi}_k - \gamma_t\boldsymbol{\phi}_t\right]. \tag{6.41}$$

Note that $\mathbf{A}_t^t$ completes the summation in $\mathbf{A}_t$ as $\mathbf{A}_t^t = \frac{1}{t}\sum_{n=0}^{t-1}\tilde{\mathbf{A}}_{n,t} = \mathbf{A}_t$.

Here, $\mathbf{A}_t^t$ is the estimate which we would like to compute strictly incrementally, and its non-strictly incremental iteration $\mathbf{A}_{k+1}^{t+1}$ uses $\mathbf{g}_{k,t+1}$ as the target.

In the following, we seek to represent this target in a similar form Y_k^t is represented in Theorem 25:

$$\mathbf{g}_{k,t+1} - \mathbf{g}_{k,t} \tag{6.42}$$

$$= \sum_{h=k+1}^{t}\gamma_{k+1}^{h-1}\lambda_{k+1}^{h-1}W_{k+1}^h\Big[(1-\gamma_h\lambda_h)\boldsymbol{\phi}_k - \gamma_h(1-\lambda_h)\boldsymbol{\phi}_h\Big] + \gamma_{k+1}^{t}\lambda_{k+1}^{t}W_{k+1}^{t+1}\left[\boldsymbol{\phi}_k - \gamma_{t+1}\boldsymbol{\phi}_{t+1}\right] \tag{6.43}$$

$$- \sum_{h=k+1}^{t-1}\gamma_{k+1}^{h-1}\lambda_{k+1}^{h-1}W_{k+1}^h\Big[(1-\gamma_h\lambda_h)\boldsymbol{\phi}_k - \gamma_h(1-\lambda_h)\boldsymbol{\phi}_h\Big] - \gamma_{k+1}^{t-1}\lambda_{k+1}^{t-1}W_{k+1}^t\left[\boldsymbol{\phi}_k - \gamma_t\boldsymbol{\phi}_t\right] \tag{6.44}$$

$$= \gamma_{k+1}\lambda_{k+1}W_{k+1}^{k+1}\Bigg[\sum_{h=k+2}^{t}\gamma_{k+2}^{h-1}\lambda_{k+2}^{h-1}W_{k+2}^h\Big[(1-\gamma_h\lambda_h)\boldsymbol{\phi}_{k+1} - \gamma_h(1-\lambda_h)\boldsymbol{\phi}_h\Big] \tag{6.45}$$

$$+ \gamma_{k+2}^{t}\lambda_{k+2}^{t}W_{k+2}^{t+1}\left[\boldsymbol{\phi}_{k+1} - \gamma_{t+1}\boldsymbol{\phi}_{t+1}\right]\Bigg] \tag{6.46}$$

$$- \gamma_{k+1}\lambda_{k+1}W_{k+1}^{k+1}\Bigg[\sum_{h=k+2}^{t-1}\gamma_{k+2}^{h-1}\lambda_{k+2}^{h-1}W_{k+2}^h\Big[(1-\gamma_h\lambda_h)\boldsymbol{\phi}_{k+1} - \gamma_h(1-\lambda_h)\boldsymbol{\phi}_h\Big] \tag{6.47}$$

$$+ \gamma_{k+2}^{t}\lambda_{k+2}^{t}W_{k+2}^{t+1}\left[\boldsymbol{\phi}_{k+1} - \gamma_t\boldsymbol{\phi}_t\right]\Bigg] + \left(W_t^{t+1}-1\right)\gamma_{k+1}^{t}\lambda_{k+1}^{t}W_{k+1}^{t}\left(\boldsymbol{\phi}_k - \boldsymbol{\phi}_{k+1}\right) \tag{6.48}$$

$$= \gamma_{k+1}\lambda_{k+1}W_{k+1}^{k+1}\left(\mathbf{g}_{k+1,t+1} - \mathbf{g}_{k+1,t}\right) + \left(W_t^{t+1} - 1\right)\gamma_{k+1}^t\lambda_{k+1}^t W_{k+1}^t\left(\boldsymbol{\phi}_k - \boldsymbol{\phi}_{k+1}\right). \quad (6.49)$$

Therefore, by letting $d_{k+1} = c_k = \gamma_{k+1}\lambda_{k+1}W_k^{k+1}$, $g_k = \gamma_{k+1}\lambda_{k+1}\left(\boldsymbol{\phi}_k - \boldsymbol{\phi}_{k+1}\right)$, and $b_t = W_t^{t+1} - 1$, we can see that $\mathbf{g}_{k,t}$ has the same recursive form as Y_k^t. Therefore, we can write $\mathbf{A}_t$ recursively as

$$\mathbf{A}_{t+1} = \left(1 - \frac{1}{t+1}\right)\mathbf{A}_t + \frac{1}{t+1}\mathbf{e}_t\left(\boldsymbol{\phi}_t - \gamma_{t+1}\boldsymbol{\phi}_{t+1}\right)^\top + \left(W_t^{t+1} - 1\right)\frac{1}{t+1}\mathbf{V}_t, \quad (6.50)$$

$$\mathbf{e}_t = W_t^{t+1}\boldsymbol{\phi}_t + \gamma_t\lambda_t W_t^{t+1}\mathbf{e}_{t-1}, \quad (6.51)$$

$$\mathbf{V}_{t+1} = \gamma_{t+1}\lambda_{t+1}\left(W_t^{t+1}\mathbf{V}_t + \mathbf{e}_t\left(\boldsymbol{\phi}_t - \boldsymbol{\phi}_{t+1}\right)^\top\right). \quad (6.52)$$

The strictly incremental update of $\mathbf{b}_t$ can also be derived in a similar way. □

6.5 Experimental Results

We compared the performance of WIS-LSTD(λ) with the conventional off-policy LSTD(λ) by Yu (2010) on two random-walk tasks for off-policy policy evaluation. These random-walk tasks consist of a Markov chain with 11 non-terminal and two terminal states. They can be imagined to be laid out horizontally, where the two terminal states are at the left and the right ends of the chain. From each non-terminal state, there are two actions available: *left*, which leads to the state to the left and *right*, which leads to the state to the right. The reward is 0 for all transitions except for the rightmost transition to the terminal state, where it is +1. The initial state was set to the state in the middle of the chain. The behavior policy chooses an action uniformly randomly, whereas the target policy chooses the *right* action with probability 0.99. The termination function γ was set to 1 for the non-terminal states and 0 for the terminal states.

We used two tasks based on this Markov chain in our experiments. These tasks differ by how the non-terminal states were mapped to features. The terminal states were always mapped to a vector with all zero elements. For each non-terminal state, the features were normalized so that the L^2 norm of each feature vector was one. For the first task, the feature representation was *tabular*, that is, the feature vectors were standard basis vectors. In this representation, each feature corresponded to only one state. For the second task, the feature vectors were binary representations of state indices. There were 11 non-terminal states, hence each feature vector had $\lfloor \log_2(11) \rfloor + 1 = 4$ components. These vectors for the states from left to right were $(0,0,0,1)^\top$, $(0,0,1,0)^\top$, $(0,0,1,1)^\top, \ldots, (1,0,1,1)^\top$, which were then normalized to get unit vectors. These features heavily underrepresented the states, because 11 states were represented by only 4 features.

We tested both algorithms for different values of constant λ, from 0 to 0.9 in steps of 0.1 and from 0.9 to 1.0 in steps of 0.025. The matrix to be inverted in both methods was initialized to $\epsilon\mathbf{I}$, where the regularization parameter ϵ was varied by powers of 10 with powers chosen from -3 to +3 in steps of 0.2. The performance was measured as the empirical

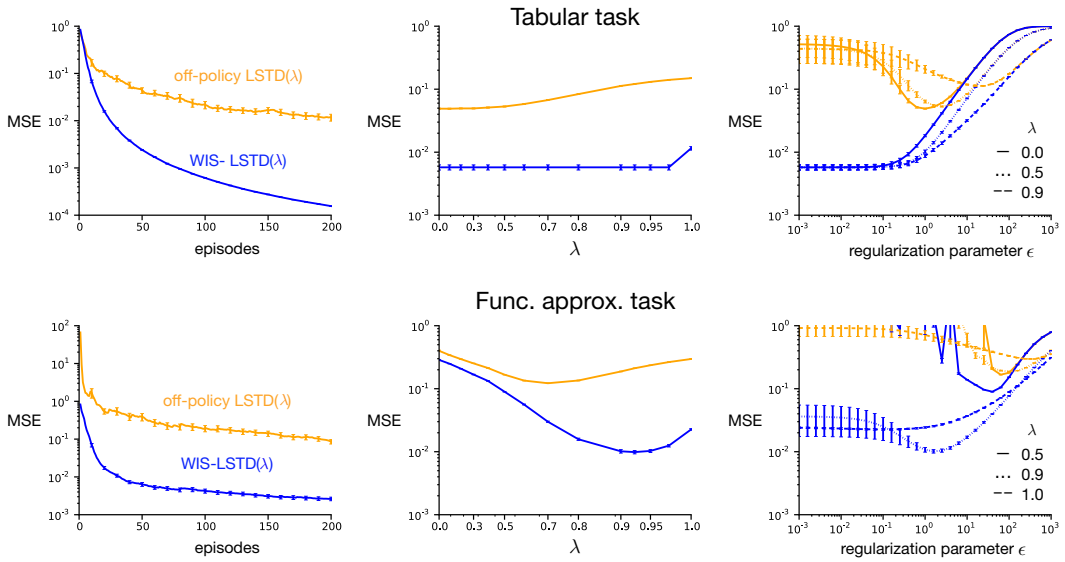

Figure 6.1: Empirical comparison of WIS-LSTD(λ) with conventional off-policy LSTD(λ) on two random-walk tasks. The empirical Mean Squared Error shown is for the initial state at the end of each episode, averaged over 100 independent runs (and also over 200 episodes in column 2 and 3).

mean squared error (MSE) between the estimated value of the initial state and its true value under the target policy projected to the space spanned by the given features. This error was measured at the end of each of 200 episodes for 100 independent runs.

Figure 1 shows the results for the two tasks in terms of empirical convergence rate, optimum performance, and parameter sensitivity. Each curve shows MSE together with standard errors. The first row shows results for the tabular task and the second row shows results for the function approximation task. The first column shows learning curves using $(\lambda, \epsilon) = (0, 1)$ for the first task and $(0.95, 10)$ for the second. It shows that in both cases WIS-LSTD(λ) learned faster and gave lower error throughout the period of learning. The second column shows performance for different λ optimized over ϵ. The x-axis is plotted in a reverse log scale, where higher values are more spread out than the lower values. In both tasks, WIS-LSTD(λ) outperformed the conventional LSTD(λ) for all values of λ. For the best parameter setting (best λ and ϵ), WIS-LSTD(λ) outperformed LSTD(λ) by an order of magnitude.

The third column shows performance with respect to different regularization parameter values ϵ for three representative values of λ. For a wide range of ϵ, WIS-LSTD(λ) outperformed conventional LSTD(λ) by an order of magnitude. Both methods performed similarly for large ϵ, as such large values essentially prevent learning for a long period. In the function approximation task when smaller values of ϵ were chosen, λ close to 1 led to more stable estimates, whereas smaller λ introduced high variance for both methods. In both tasks, the better-performing regions of ϵ (the U-shaped depressions) were wider for

WIS-LSTD(λ).

6.6 Conclusions

In this chapter, we developed the fullest extension of weighted importance sampling to the case of linear function approximation with bootstrapping and real-time strictly incremental update. The resulting algorithm is WIS-LSTD(λ), which we have shown to carry over the benefits of the tabular off-policy estimator WIS2, which we found to be superior among other estimators. Development of this algorithm required some important contributions toward the understanding of reinforcement learning algorithms. We developed a technique for incorporating state-dependent bootstrapping to combine different corrected flat returns. To make real-time updates, we developed the concept of interim targets, so that meaningful updates can be made without waiting for a full termination. We developed a general equivalence technique for mechanistically deriving strictly incremental updates of reinforcement learning algorithm, including WIS-LSTD(λ) we developed here. This is the first off-policy algorithm for function approximation that fully incorporates the benefits of weighted importance sampling. One drawback of this algorithm is that it is a least-squares method, requiring matrix inversion or at least $O(m^2)$ memory and computation. Due to strictly incremental computation, it can still be used for many off-policy predictions. However, as White (2015) demonstrated, stochastic approximation algorithms with linear computational complexity are much more suited for large-scale learning of predictions compared to least-squares methods.

Chapter 7

Weighted Importance Sampling with Linear Computational Complexity [1]

In this chapter, we introduce a class of new off-policy algorithms with linear computational complexity based on weighted importance sampling. They constitute our final contribution toward the issue of high variance based on weighted importance sampling. These algorithms emerged from our attempt to retain the benefits of WIS-LSTD(λ) while reducing the computational complexity. Our effort also resulted into a novel way of incorporating averaging methods with parametric function approximation, which constitute another contribution of this thesis.

To produce computationally cheap algorithms based on weighted importance sampling, we utilize stochastic gradient descent (SGD) updates as they typically fulfill this computational requirement. To carry over weighted importance sampling, which is a tabular averaging method to stochastic gradient descent updates, we seek a principled way of relating tabular averaging method with stochastic gradient descent with function approximation. Therefore, the key step in this endeavor is bridging the gap between stochastic gradient descent updates with parametric function approximation and averaging methods with lookup table representation. We show that this gap can be bridged by keeping track of the "usage" of each feature and use it to modulate the step size of the updates. Although weighted importance sampling based on stochastic gradient descent with function approximation can be achieved through the application of a feature *usage* vector, it could not be calculated in a strictly incremental manner. We introduce a second stochastic gradient descent update which is motivated by weighted importance sampling and can be computed in a strictly incremental manner. Moreover, we show that the feature usage vector can be flexibly applied to other existing algorithms to achieve reduced mean squared errors.

[1] This chapter is adapted from a published paper coauthored by this author (Mahmood & Sutton 2015).

7.1 Merging Sample Average and SGD

In this section, we investigate the relationship between the sample average estimator and SGD. We first show that SGD does not reduce to the sample average estimator in the fully-representable case also known as the *tabular* representation. Then we propose a modification to SGD and show that it achieves the sample average estimator when the feature representation is tabular.

The sample average is one of the simplest Monte Carlo estimators. In order to introduce it, consider that data arrives as a sequence of samples $Y_k \in \mathbb{R}$ drawn from a fixed distribution. The goal of the learner is to estimate the expected value of the samples, $v \stackrel{\text{def}}{=\!=} \mathrm{E}[Y_k]$. The sample average estimator $\hat{V}_{t+1}$ for data given up to time t can be defined and incrementally updated in the following way:

$$\hat{V}_{t+1} \stackrel{\text{def}}{=\!=} \frac{\sum_{k=1}^{t} Y_k}{t} = \hat{V}_t + \frac{1}{t}\left(Y_t - \hat{V}_t\right);\ \hat{V}_1 \stackrel{\text{def}}{=\!=} 0. \tag{7.1}$$

In the incremental update, $\frac{1}{t}$ can be viewed as a form of step size, modulating the amount of change made to the current estimate, which decreases with time in this case. In the parametric function approximation case, we have to go beyond sample average and use stochastic approximation methods such as SGD.

To introduce SGD, we use a supervised-learning setting with linear function approximation. In this setting, data arrives as a sequence of input-output pairs (X_k, Y_k), where X_k takes values from a finite set $\mathcal{X}$ and $Y_k \in \mathbb{R}$. The learner observes a feature representation of the inputs, where each input is mapped to a feature vector $\boldsymbol{\phi}_k \stackrel{\text{def}}{=\!=} \boldsymbol{\phi}(X_k) \in \mathbb{R}^m$. The goal of the learner is to estimate the conditional expectation of Y_k for each unique input $\boldsymbol{\phi} \in \mathcal{X}$ as a linear function of the features: $\boldsymbol{\theta}^\top \boldsymbol{\phi}(x) \approx v(\boldsymbol{\phi}) \stackrel{\text{def}}{=\!=} \mathrm{E}[Y_k | X_k = \boldsymbol{\phi}]$. SGD incrementally updates the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^m$ at each time step t in the following way:

$$\boldsymbol{\theta}_{t+1} \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_t + \alpha_t \left(Y_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) \boldsymbol{\phi}_t, \tag{7.2}$$

where $\alpha_t > 0$ is a scalar step-size parameter, which is often set to a small constant. The per-update time and memory complexity of SGD is $O(n)$.

Linear function approximation includes tabular representations as a special case. For example, if the feature vectors are $|\mathcal{X}|$-dimensional standard basis vectors, then each feature uniquely represents an input, and the feature representation becomes tabular.

We are interested in finding whether SGD degenerates to sample average when the linear function approximation setting reduces to the tabular setting. Both incremental updates are in a form where the previous estimate is incremented with a product of an error and a step size. In the SGD update, the product also has the feature vector as a factor, but in the tabular setting, it simply selects the input for which an update is made.

A major difference between SGD and sample average is the ability of SGD to track under non-stationarity through the use of a constant step size. Typically, the step size of

SGD is set to a constant value or decreased with time, where the latter does not work well under non-stationarity but is similar to how sample average works. While we attempt to accommodate sample average estimation more closely within SGD, it is also desirable to retain the tracking ability of SGD.

SGD clearly cannot achieve sample average with a constant step size. On the other hand, if we set the step-size parameter in the SGD update as $\alpha_t = \frac{1}{t}$, the SGD update still does not subsume the sample average. This is because, in the SGD update (7.2), time t is the total number of samples seen so far, whereas, in the sample average update (7.1), it is the number of samples seen so far only for one specific input.

We take two important steps to bridge the gap between the sample average update and the SGD update. First, we extend the sample average estimator to incorporate tracking through recency weighting, where the amount of weight assigned to the recent samples is modulated by a scalar recency-weighting constant. This new *recency-weighted average* estimator subsumes sample average as a special case and hence unifies both tracking and sample averaging. Second, we propose a variant of SGD that reduces to recency-weighted average in the tabular setting and still uses only $O(n)$ per-update memory and computation.

Our proposed recency-weighted average estimator can be derived by minimizing an empirical mean squared objective with recency weighting:

$$\tilde{V}_{t+1} \stackrel{\text{def}}{=\!=} \arg\min_v \frac{1}{t}\sum_{k=1}^{t}(1-\eta)^{t-k}\left(Y_k - v\right)^2; 0 \leq \eta < 1.$$

Here, the recency-weighting constant η exponentially weights the past observations down and thus gives more weight to the recent samples. When $\eta = 0$, all samples are weighted equally. The recency-weighted average can be defined and incrementally updated as follows:

$$\tilde{V}_{t+1} = \frac{\sum_{k=1}^{t}(1-\eta)^{t-k}Y_k}{\sum_{k=1}^{t}(1-\eta)^{t-k}} = \tilde{V}_t + \frac{1}{\tilde{U}_{t+1}}\Big(Y_t - \tilde{V}_t\Big), \tag{7.3}$$

$$\tilde{U}_{t+1} \stackrel{\text{def}}{=\!=} (1-\eta)\tilde{U}_t + 1; \qquad \tilde{U}_1 \stackrel{\text{def}}{=\!=} 0, \quad \tilde{V}_1 \stackrel{\text{def}}{=\!=} 0. \tag{7.4}$$

It is easy to see that the recency-weighted average is an unbiased estimator of v. Moreover, when $\eta = 0$, it reduces to the sample average estimator.

Now, we propose a modified SGD in the supervised-learning setting that for tabular representation reduces to the recency-weighted average. The updates are as follows:

$$\mathbf{u}_{t+1} \stackrel{\text{def}}{=\!=} (\mathbf{1} - \eta\boldsymbol{\phi}_t \circ \boldsymbol{\phi}_t) \circ \mathbf{u}_t + \boldsymbol{\phi}_t \circ \boldsymbol{\phi}_t, \tag{7.5}$$

$$\boldsymbol{\alpha}_{t+1} \stackrel{\text{def}}{=\!=} \mathbf{1} \oslash \mathbf{u}_{t+1}, \tag{7.6}$$

$$\boldsymbol{\theta}_{t+1} \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_t + \boldsymbol{\alpha}_{t+1} \circ \Big(Y_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\Big)\boldsymbol{\phi}_t, \tag{7.7}$$

where $\eta \geq 0$ is the recency-weighting factor, $\circ$ is component-wise vector multiplication, $\oslash$ is component-wise vector division where a division by zero results in zero, and $\mathbf{1} \in \mathbb{R}^m$ is a

vector of all ones. Here, $\boldsymbol{\alpha}_{t+1} \in \mathbb{R}^m$ is a vector step-size parameter, set as the vector division of $\mathbf{1}$ by $\mathbf{u}_{t+1} \in \mathbb{R}^m$, which parallels $\tilde{U}$ of the recency-weighted average. We call $\mathbf{u}$ the *usage vector*, as it can be seen as an estimate of how much each feature is "used" over time by the update. We call this algorithm the *usage-based SGD* (U-SGD). Replacing a division by zero with zero in the step-size vector amounts to having no updates for the corresponding component. This makes sense because a zero in any component of $\mathbf{u}$ can occur only at the beginning when $\mathbf{u}$ is initialized to zero and the corresponding feature has not been activated yet. Once a feature is nonzero, the corresponding component of $\boldsymbol{\alpha}$ becomes positive and, with sufficiently small η, it always remains so.

In the following theorem, we show that U-SGD reduces to recency-weighted average in the tabular setting and hence is a generalization of the sample average estimator as well.

Theorem 27 (Backward consistency of U-SGD with sample average)**.** *If the feature representation is tabular, the vectors* $\mathbf{u}$ *and* $\boldsymbol{\theta}$ *are initially set to zero, and* $0 \leq \eta < 1$*, then U-SGD defined by (7.5)-(7.7) degenerates to the recency-weighted average estimator defined by (7.3) and (7.4), in the sense that each component of the parameter vector* $\boldsymbol{\theta}_{t+1}$ *of U-SGD becomes the recency-weighted average estimator of the corresponding input.*

Proof. Consider that t samples have been observed and among them t_x samples correspond to input $\boldsymbol{\phi}$. Hence, $\sum_{\boldsymbol{\phi} \in \mathcal{X}} t_x = t$. Let $Y_{\boldsymbol{\phi},k}$ denote the kth output corresponding to input $\boldsymbol{\phi}$. Then the recency-weighted average estimator of $v(x)$ given overall data up to t can be equivalently redefined in the following way:

$$\begin{aligned}
\tilde{V}_{t_x+1} &\overset{\text{def}}{=\!=} \frac{\sum_{k=1}^{t_x} (1-\eta)^{t_x-k} Y_{\boldsymbol{\phi},k}}{\sum_{k=1}^{t_x} (1-\eta)^{t_x-k}} \\
&= \tilde{V}_{t_x} + \frac{1}{\tilde{U}_{t_x+1}} \left(Y_{x,t_x} - \tilde{V}_{t_x} \right); \quad \tilde{V}_1 = 0, \\
\tilde{U}_{t_x+1} &\overset{\text{def}}{=\!=} (1-\eta)\tilde{U}_{t_x} + 1; \qquad\qquad \tilde{U}_1 = 0.
\end{aligned}$$

Consider that the ith feature corresponds to input $\boldsymbol{\phi}$. Then it is equivalent to prove that $[\boldsymbol{\theta}_{t+1}]_i = \tilde{V}_{t_x+1}$, where $[\cdot]_i$ denotes the ith component of a vector.

We prove by induction. First we show that $[\mathbf{u}_{t+1}]_i = \tilde{U}_{t_x+1}$. By assumption, $[\mathbf{u}_1]_i = \tilde{U}_1 = 0$. Now, consider that $[\mathbf{u}_t]_i = \tilde{U}_{(t-1)_x+1}$. Then the ith component of $\mathbf{u}_{t+1}$ can be written as

$$[\mathbf{u}_{t+1}]_i = (1 - \eta[\boldsymbol{\phi}_t]_i^2)[\mathbf{u}_t]_i + [\boldsymbol{\phi}_t]_i^2.$$

If the tth input is not x, then $t_x = (t-1)_x$ and $[\boldsymbol{\phi}_t]_i = 0$. Hence

$$[\mathbf{u}_{t+1}]_i = (1-0)\tilde{U}_{(t-1)_x+1} + 0 = \tilde{U}_{(t-1)_x+1} = \tilde{U}_{t_x+1}.$$

On the other hand, if the tth input is x, then $t_x = (t-1)_x + 1$ and $[\boldsymbol{\phi}_t]_i = 1$. Hence,

$$[\mathbf{u}_{t+1}]_i = (1-\eta)\tilde{U}_{(t-1)_x+1} + 1 = (1-\eta)\tilde{U}_{t_x} + 1 = \tilde{U}_{t_x+1}.$$

Hence, $[\boldsymbol{\alpha}_{t+1}]_i = \frac{1}{\tilde{U}_{t_x+1}}$, if $t_x > 0$, or $[\boldsymbol{\alpha}_{t+1}]_i = 0$, otherwise.

Now, by assumption, $[\boldsymbol{\theta}_1]_i = \tilde{V}_1 = 0$. Consider $[\boldsymbol{\theta}_t]_i = \tilde{V}_{(t-1)_x+1}$ and $t_x > 0$. Then the ith component of $\boldsymbol{\theta}_{t+1}$ can be written as

$$\begin{aligned}[\boldsymbol{\theta}_{t+1}]_i &= [\boldsymbol{\theta}_t]_i + [\boldsymbol{\alpha}_{t+1}]_i \left(Y_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) [\boldsymbol{\phi}_t]_i \\ &= \tilde{V}_{(t-1)_x+1} + \frac{1}{\tilde{U}_{t_x+1}} \left(Y_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) [\boldsymbol{\phi}_t]_i.\end{aligned}$$

If the tth input is not x, then $[\boldsymbol{\theta}_{t+1}]_i = \tilde{V}_{(t-1)_x+1} + 0 = \tilde{V}_{t_x+1}$.

On the other hand, if the tth input is x, then $Y_t = Y_{\boldsymbol{\phi},t_x}$ and

$$\begin{aligned}[\boldsymbol{\theta}_{t+1}]_i &= \tilde{V}_{(t-1)_x+1} + \frac{1}{\tilde{U}_{t_x+1}} \left(Y_{\boldsymbol{\phi},t_x} - \tilde{V}_{(t-1)_x+1}\right) \\ &= \tilde{V}_{t_x} + \frac{1}{\tilde{U}_{t_x+1}} \left(Y_{\boldsymbol{\phi},t_x} - \tilde{V}_{t_x}\right) = \tilde{V}_{t_x+1}.\end{aligned}$$

The only case that is left is when $t_x = 0$. In this case, the tth input cannot be x, and $\tilde{V}_{t_x+1} = \tilde{V}_{(t-1)_x+1} = \cdots = \tilde{V}_1 = 0$. Then

$$\begin{aligned}[\boldsymbol{\theta}_{t+1}]_i &= [\boldsymbol{\theta}_t]_i + [\boldsymbol{\alpha}_{t+1}]_i \left(Y_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) [\boldsymbol{\phi}_t]_i \\ &= \tilde{V}_{(t-1)_x+1} + 0 \cdot \left(Y_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) \cdot 0 \\ &= 0 = \tilde{V}_{t_x+1}.\end{aligned}$$

□

7.2 Merging WIS and off-policy SGD

In this section, we carry over weighted importance sampling (WIS) to off-policy SGD, drawing from the ideas developed in the previous section. We introduce two new off-policy SGD algorithms based on WIS. The first one subsumes WIS fully but does not lead to an $O(n)$ implementation, whereas the other algorithm is more amenable to an efficient implementation.

First we introduce both the ordinary importance sampling (OIS) and WIS. Importance sampling is a technique for estimating an expectation under one distribution using samples drawn from a different distribution. OIS estimates the expectation by forming a special kind of sample average. Consider that samples $Y_k \in \mathbb{R}$ are drawn from a sample distribution l, but the goal of the learner is to estimate the expectation $v_g \stackrel{\text{def}}{=\!=} \mathrm{E}_g[Y_k]$ under a different distribution g. OIS estimates v_g by scaling each sample Y_k by the importance-sampling ratio $W_k \stackrel{\text{def}}{=\!=} \frac{g(Y_k)}{l(Y_k)}$ and forming a sample average estimate of the scaled samples:

$$\tilde{V}_{t+1} \stackrel{\text{def}}{=\!=} \frac{\sum_{k=1}^{t} W_k Y_k}{t} = \tilde{V}_t + \frac{1}{t}\left(W_t Y_t - \tilde{V}_t\right); \quad \tilde{V}_1 \stackrel{\text{def}}{=\!=} 0.$$

WIS, on the other hand, estimates v_g by forming a weighted average estimate of the original samples. Its definition and incremental update are as follows:

$$\hat{V}_{t+1} \stackrel{\text{def}}{=\!=} \frac{\sum_{k=1}^{t} W_k Y_k}{\sum_{k=1}^{t} W_k} = \hat{V}_t + \frac{1}{\hat{U}_{t+1}} W_t \left(Y_t - \hat{V}_t\right),$$
$$\hat{U}_{t+1} \stackrel{\text{def}}{=\!=} \hat{U}_t + W_t; \qquad \hat{U}_1 \stackrel{\text{def}}{=\!=} 0, \quad \hat{V}_1 \stackrel{\text{def}}{=\!=} 0.$$

If there is no discrepancy between the sample and the target distribution, then $W_k = 1, \forall k$, and both OIS and WIS become equivalent to the sample average estimator.

We derive the *recency-weighted WIS* as a solution to a mean squared objective with recency weighting and additionally importance sampling:

$$\bar{V}_{t+1} \stackrel{\text{def}}{=\!=} \arg\min_v \frac{1}{t} \sum_{k=1}^{t} (1-\eta)^{t-k} W_k \left(Y_k - v\right)^2; 0 \leq \eta < 1,$$
$$= \frac{\sum_{k=1}^{t} (1-\eta)^{t-k} W_k Y_k}{\sum_{k=1}^{t} (1-\eta)^{t-k} W_k}.$$

It is easy to see that, when $\eta = 0$, the recency-weighted WIS estimator reduces to WIS. Recency-weighted WIS can be updated incrementally in the following way:

$$\bar{V}_{t+1} = \bar{V}_t + \frac{1}{\mathbf{d}_{t+1}} W_t \left(Y_t - \bar{V}_t\right); \quad \bar{V}_1 \stackrel{\text{def}}{=\!=} 0, \tag{7.8}$$
$$\mathbf{d}_{t+1} \stackrel{\text{def}}{=\!=} (1-\eta)\mathbf{d}_t + W_t; \qquad \mathbf{d}_1 \stackrel{\text{def}}{=\!=} 0. \tag{7.9}$$

Now we introduce two variants of SGD based on WIS in a more general off-policy reinforcement learning setting with linear function approximation.

We call the first off-policy SGD based on WIS to be *WIS-SGD-1*. With $0 \leq k < t+1$ and $\boldsymbol{\theta}_0^t \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_0, \forall t$, the following updates define WIS-SGD-1:

$$\mathbf{u}_{k+1}^{t+1} \stackrel{\text{def}}{=\!=} (\mathbf{1} - \eta \boldsymbol{\phi}_k \circ \boldsymbol{\phi}_k) \circ \mathbf{u}_k^{t+1} + \rho_k^{t+1} \boldsymbol{\phi}_k \circ \boldsymbol{\phi}_k, \tag{7.10}$$
$$\boldsymbol{\alpha}_{k+1}^{t+1} \stackrel{\text{def}}{=\!=} \mathbf{1} \oslash \mathbf{u}_{k+1}^{t+1}, \tag{7.11}$$
$$\boldsymbol{\theta}_{k+1}^{t+1} \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_k + \boldsymbol{\alpha}_{k+1}^{t+1} \circ \rho_k^{t+1} \left(G_k^{t+1} - \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^{t+1}\right) \boldsymbol{\phi}_k. \tag{7.12}$$

Similar to U-SGD, WIS-SGD-1 maintains a vector step size through the update of a usage vector, which in this case also includes the importance-sampling ratios. Unlike U-SGD, the parameters of WIS-SGD-1 use two-time indices. The time index in the subscript corresponds to the time step of the prediction, and the time index in the superscript stands for the data horizon. In the following, we show that WIS-SGD-1 reduces to recency-weighted WIS, and hence to WIS as well, in the tabular setting.

Theorem 28 (Backward consistency of WIS-SGD-1 with WIS)**.** *If the feature representation is tabular, the vectors* $\mathbf{u}$ *and* $\boldsymbol{\theta}$ *are initially set to zero, and* $0 \leq \eta < 1$*, then WIS-SGD-1 defined by (7.10)-(7.12) degenerates to recency-weighted WIS defined by (7.8) and (7.9) with* $Y_k \stackrel{\text{def}}{=\!=} G_k^{t+1}$ *and* $W_k \stackrel{\text{def}}{=\!=} \rho_k^{t+1}$*, in the sense that each component of*

the parameter vector $\boldsymbol{\theta}_{t+1}^{t+1}$ of WIS-SGD-1 becomes the recency-weighted WIS estimator of the corresponding input.

Proof. The proof is similar to that of Theorem 27.

Consider that data is available up to time $t+1$, among which state s was visited on t_s steps. Let $G_{s,k}^{t+1}$ denote the kth flat truncated return originated from state s and $\rho_{s,k}^{t+1}$ its corresponding importance-sampling ratio. Then the recency-weighted WIS estimator of $v(s)$ given overall data up to $t+1$ can be equivalently redefined in the following way:

$$\bar{V}_{t_s+1}^{t+1} \overset{\text{def}}{=\!=} \bar{V}_{t_s}^{t+1} + \frac{\rho_{s,t_s}^{t+1}}{\mathbf{d}_{t_s+1}^{t+1}} \left(G_{s,t_s}^{t+1} - \bar{V}_{t_s}^{t+1}\right); \quad \bar{V}_0^{t+1} = 0,$$
$$\mathbf{d}_{t_s+1}^{t+1} \overset{\text{def}}{=\!=} (1-\eta)\mathbf{d}_{t_s}^{t+1} + \rho_{s,t_s}^{t+1}; \qquad\qquad \mathbf{d}_0^{t+1} = 0.$$

Consider that the ith feature corresponds to input s. Then it is equivalent to prove that $\left[\boldsymbol{\theta}_{t+1}^{t+1}\right]_i = \bar{V}_{t_s+1}^{t+1}$, where $[\cdot]_i$ denotes the ith component of a vector. By abuse of notation, we drop all the $t+1$ from superscripts, as it is redundant in this proof.

We prove by induction. First we show that $[\mathbf{u}_{t+1}]_i = \mathbf{d}_{t_s+1}$. By assumption, $[\mathbf{u}_0]_i = \mathbf{d}_0 = 0$. Considering $[\mathbf{u}_t]_i = \mathbf{d}_{(t-1)_s+1}$. Then the ith component of $\mathbf{u}_{t+1}$ can be written as

$$[\mathbf{u}_{t+1}]_i = (1 - \eta[\boldsymbol{\phi}_t]_i^2)[\mathbf{u}_t]_i + \rho_t[\boldsymbol{\phi}_t]_i^2.$$

If the state at time t is not s, then $t_s = (t-1)_s$ and $[\boldsymbol{\phi}_t]_i = 0$. Hence

$$[\mathbf{u}_{t+1}]_i = (1-0)\mathbf{d}_{(t-1)_s+1} + 0 = \mathbf{d}_{(t-1)_s+1} = \mathbf{d}_{t_s+1}.$$

On the other hand, if the state at time t is s, then $t_s = (t-1)_s + 1$, $[\boldsymbol{\phi}_t]_i = 1$ and $\rho_t = \rho_{s,t_s}^{t+1}$. Hence,

$$\begin{aligned}[\mathbf{u}_{t+1}]_i &= (1-\eta)\mathbf{d}_{(t-1)_s+1} + \rho_{s,t_s}^{t+1} \\ &= (1-\eta)\mathbf{d}_{t_s} + \rho_{s,t_s}^{t+1} = \mathbf{d}_{t_s+1}.\end{aligned}$$

Hence, $[\boldsymbol{\alpha}_{t+1}]_i = \frac{1}{\mathbf{d}_{t_s+1}}$, if $t_s > 0$, or $[\boldsymbol{\alpha}_{t+1}]_i = 0$, otherwise.

Now, by assumption, $[\boldsymbol{\theta}_0]_i = \bar{V}_0 = 0$. Considering $[\boldsymbol{\theta}_t]_i = \bar{V}_{(t-1)_s+1}$ and $t_s > 0$, the ith component of $\boldsymbol{\theta}_{t+1}$ can be written as

$$\begin{aligned}[\boldsymbol{\theta}_{t+1}]_i &= [\boldsymbol{\theta}_t]_i + [\boldsymbol{\alpha}_{t+1}]_i \rho_t \left(G_t - \boldsymbol{\phi}_t^\top \boldsymbol{\theta}_t\right) [\boldsymbol{\phi}_t]_i \\ &= \bar{V}_{(t-1)_s+1} + \frac{\rho_t}{\mathbf{d}_{t_s+1}} \left(G_t - \boldsymbol{\phi}_t^\top \boldsymbol{\theta}_t\right) [\boldsymbol{\phi}_t]_i.\end{aligned}$$

If the state at time t is not s, then $[\boldsymbol{\theta}_{t+1}]_i = \bar{V}_{(t-1)_s+1} + 0 = \bar{V}_{t_s+1}$.

If the state at time t is not s, then $\rho_t = \rho_{s,t_s}$, $G_t = G_{s,t_s}$ and

$$\begin{aligned}[\boldsymbol{\theta}_{t+1}]_i &= \bar{V}_{(t-1)_s+1} + \frac{\rho_{s,t_s}}{\mathbf{d}_{t_s+1}} \left(G_{s,t_s} - \bar{V}_{(t-1)_s+1}\right) \\ &= \bar{V}_{t_s} + \frac{\rho_{s,t_s}}{\mathbf{d}_{t_s+1}} \left(G_{s,t_s} - \bar{V}_{t_s}\right) = \bar{V}_{t_s+1}.\end{aligned}$$

The only case that is left is when $t_s = 0$. In this case, the the state at time t cannot be s, and $\bar{V}_{t_s+1} = \bar{V}_{(t-1)_s+1} = \cdots = \bar{V}_0 = 0$. Then

$$\begin{aligned}
[\boldsymbol{\theta}_{t+1}]_i &= [\boldsymbol{\theta}_t]_i + [\boldsymbol{\alpha}_{t+1}]_i \rho_t \left(G_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) [\boldsymbol{\phi}_t]_i \\
&= \bar{V}_{(t-1)_s+1} + 0 \cdot \rho_t \left(G_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) \cdot 0 \\
&= 0 = \bar{V}_{t_s+1}.
\end{aligned}$$

□

Now we focus on whether and how WIS-SGD-1 can be implemented efficiently. The updates as defined above cannot be computed in $O(n)$ per time step. An update for step k requires computing an importance-sampling ratio and a flat truncated return that are available only at $t+1 > k$. It can be computed by looking ahead into the future from k, but then the update becomes acausal. It can alternatively be computed by waiting until time $t+1$ and iterating for each k. But then the update made at $t+1$ becomes expensive, scaling linearly with t, that is, $O(tn)$.

Such updates, where samples are available in future from the time step when the update is made, are often known as *forward-view* updates (Sutton & Barto 1998). Forward-view updates are typically expensive, but for some forward-view updates it is possible to derive causal and efficient updates, known as *backward-view* updates, that compute exactly the same estimate at each time step. Classically these equivalences were achieved for offline updating. Van Seijen and Sutton (2014) showed that such equivalences can also be achieved in the online case.

Converting a forward-view update into an efficient backward-view update depends on combining the extra data available at $t+1$ with the current estimate $\boldsymbol{\theta}_t^t$ in an efficient way to give the next estimate $\boldsymbol{\theta}_{t+1}^{t+1}$. For linear recursive updates, it is tantamount to unrolling both $\boldsymbol{\theta}_{t+1}^{t+1}$ and $\boldsymbol{\theta}_t^t$ and expressing their difference in a form that can be computed efficiently. It is often not possible to achieve such efficient backward-view updates, and we believe WIS-SGD-1 is one such case.

To appreciate why an efficient backward-view update of WIS-SGD-1 is not plausible, consider the update of $\boldsymbol{\theta}_t^t$ unrolled back to the beginning of time:

$$\begin{aligned}
\boldsymbol{\theta}_t^t &= \left(\mathbf{I} - \rho_{t-1}^t (\boldsymbol{\alpha}_t^t \circ \boldsymbol{\phi}_{t-1}) \boldsymbol{\phi}_{t-1}^\top\right) \boldsymbol{\theta}_{t-1}^t + \rho_{t-1}^t G_{t-1}^t \boldsymbol{\phi}_{t-1} \\
&= \prod_{k=0}^{t-1} \left(\mathbf{I} - \rho_k^t (\boldsymbol{\alpha}_{k+1}^t \circ \boldsymbol{\phi}_k) \boldsymbol{\phi}_k^\top\right) \boldsymbol{\theta}_0^t \\
&\quad + \sum_{k=0}^{t-1} \rho_k^t G_k^t \prod_{j=k+1}^{t-1} \left(\mathbf{I} - \rho_j^t (\boldsymbol{\alpha}_{j+1}^t \circ \boldsymbol{\phi}_j) \boldsymbol{\phi}_j^\top\right) \boldsymbol{\phi}_k.
\end{aligned}$$

In order to obtain $\boldsymbol{\theta}_{t+1}^{t+1}$ by combining the new data $\boldsymbol{\phi}_t$ and R_{t+1} with $\boldsymbol{\theta}_t^t$, it is evident that each of the $\rho_k^t (\boldsymbol{\alpha}_{k+1}^t \circ \boldsymbol{\phi}_k) \boldsymbol{\phi}_k^\top$ terms in the first product needs to be replaced by $\rho_k^{t+1} (\boldsymbol{\alpha}_{k+1}^{t+1} \circ$

$\boldsymbol{\phi}_k)\boldsymbol{\phi}_k^\top$, which is unlikely to be achieved in an inexpensive way. We cannot achieve a backward-view due to the lack of the distributivity of addition over multiplication as in (3.55). This problem does not appear in previous algorithms with online equivalence such as true online TD(λ) (van Seijen & Sutton 2014) or true online GTD(λ) (van Hasselt, Mahmood & Sutton 2014), because the terms involved in the product of the unrolled update in those algorithms do not involve *forward-view* terms, that is, they contain ρ_k and α_{k+1} in those products instead of ρ_k^{t+1} and $\boldsymbol{\alpha}_{k+1}^{t+1}$. This specific problem with WIS-SGD-1 is due to the fact that the error of the update in (7.12) is multiplied by the forward-view terms ρ_k^{t+1} and $\boldsymbol{\alpha}_{k+1}^{t+1}$.

The observation we made in the above leads us to develop a second off-policy SGD. In this algorithm, first we replace the forward-view term $\boldsymbol{\alpha}_{k+1}^{t+1}$ from the update of $\boldsymbol{\theta}$ with $\boldsymbol{\alpha}_{k+1}^{k+1}$. Second, instead of multiplying the terms in the error $G_k^{t+1} - \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^{t+1}$ with the same forward-view term ρ_k^{t+1}, we multiply the first term G_k^{t+1} by ρ_k^{t+1} and the second term $\boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^{t+1}$ by ρ_k. To account for this discrepancy, we add two more terms in the error, and the resultant error of the new update becomes $\rho_k^{t+1} G_k^{t+1} - \rho_k^{t+1} \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1}^{k-1} + \rho_k \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1}^{k-1} - \rho_k \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^{t+1}$. Here, the first two terms are approximating the WIS-SGD-1 error $\rho_k^{t+1}\left(G_k^{t+1} - \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^{t+1}\right)$, whereas the last two terms are adding a bias. Although this new algorithm no longer reduces to WIS in the tabular setting, it is developed based on WIS and still retains the main ideas behind recency-weighted WIS. Hence, we call this algorithm *WIS-SGD-2*. The following updates define WIS-SGD-2, with $0 \leq k < t+1$:

$$\mathbf{u}_{k+1}^{t+1} \stackrel{\text{def}}{=\!=} (\mathbf{1} - \eta \boldsymbol{\phi}_k \circ \boldsymbol{\phi}_k) \circ \mathbf{u}_k^{t+1} + \rho_k^{t+1} \boldsymbol{\phi}_k \circ \boldsymbol{\phi}_k, \tag{7.13}$$

$$\boldsymbol{\alpha}_{k+1} \stackrel{\text{def}}{=\!=} \mathbf{1} \oslash \mathbf{u}_{k+1}^{k+1}, \tag{7.14}$$

$$\begin{aligned} \delta_k^{t+1} \stackrel{\text{def}}{=\!=}\ & \rho_k^{t+1} G_k^{t+1} - \rho_k^{t+1} \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1} \\ & + \rho_k \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1} - \rho_k \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^{t+1}, \end{aligned} \tag{7.15}$$

$$\boldsymbol{\theta}_{k+1}^{t+1} \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_k^{t+1} + \boldsymbol{\alpha}_{k+1} \circ \delta_k^{t+1} \boldsymbol{\phi}_k. \tag{7.16}$$

Here, $\boldsymbol{\theta}_k \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_k^k$, and $\boldsymbol{\theta}_{-1} = \mathbf{0}$. It can be easily verified that, in the on-policy case, WIS-SGD-2 degenerates to U-SGD and hence retains the backward consistency with the sample average estimator.

Although this algorithm has much more plausibility of having an efficient backward view due to the careful modifications, it is not yet immediately clear how such a backward-view update can be obtained. Van Hasselt, Mahmood, and Sutton (2014) introduced an online equivalence technique from which both true online TD(λ) and true online GTD(λ) can be derived. Their technique requires the target in error to have a specific recurrence relation. Unfortunately, that specific relation does not hold for the target in WIS-SGD-2. A new technique is needed to derive an efficient backward view for WIS-SGD-2.

7.3 Usage-based Algorithms

In this section, we develop a new off-policy algorithm that generalizes WIS-SGD-2 to partial termination and bootstrapping. Then we use the new online equivalence technique to derive an equivalent $O(n)$ backward-view update. We use a state-dependent bootstrapping parameter $\lambda_k \stackrel{\text{def}}{=\!=} \lambda(S_k) \in [0, 1]$ in developing the new algorithm. First, we construct the target, and then we define a new update for the usage vector $\mathbf{u}$ in this more general setting.

Based on the general off-policy forward view by Sutton et al. (2014), we combine truncated returns G_k^{t+1} and truncated corrected returns $G_k^{t+1} + \boldsymbol{\phi}_{t+1}^\top \boldsymbol{\theta}_t$ scaled by corresponding weights due to discounting, bootstrapping and importance sampling to develop an overall return:

$$\begin{aligned} G_{k,t+1}^{\rho} \stackrel{\text{def}}{=\!=} \rho_k C_k^t \Big((1-\gamma_{t+1}) G_k^{t+1} + \gamma_{t+1} \left(G_k^{t+1} + \boldsymbol{\phi}_{t+1}^\top \boldsymbol{\theta}_t \right) \Big) \\ + \sum_{i=k+1}^{t} \rho_k C_k^{i-1} \Big((1-\gamma_i) G_k^i + \gamma_i (1-\lambda_i) \left(G_k^i + \boldsymbol{\phi}_i^\top \boldsymbol{\theta}_{i-1} \right) \Big) \\ - \rho_k \left(C_k^t + \sum_{i=k+1}^{t} C_k^{i-1} (1 - \gamma_i \lambda_i) - 1 \right) \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1}, \end{aligned} \tag{7.17}$$

where $C_k^t \stackrel{\text{def}}{=\!=} \prod_{j=k+1}^{t} \gamma_j \lambda_j \rho_j$, $\boldsymbol{\theta}_k \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_k^k$, $0 \leq k < t+1$ and $\boldsymbol{\theta}_{-1} = \mathbf{0}$. It can be readily verified that, when no bootstrapping is used, that is, $\lambda_k = 1, \forall k$ and discounting occurs only at the data horizon $t+1$, that is, $\gamma_0 = \gamma_1 = \cdots = \gamma_t = 1$ and $\gamma_{t+1} = 0$, then $G_{k,t+1}^{\rho} = \rho_k^{t+1} G_k^{t+1} - \rho_k^{t+1} \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1} + \rho_k \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_{k-1}$. Hence $G_{k,t+1}^{\rho}$ is a strict generalization of the WIS-SGD-2 target to the state-dependent discounting and bootstrapping.

The usage vector $\mathbf{u}$ of the WIS-SGD algorithms rescales the components of the parameter updates to clamp down the updates proportionally when they become large due to large importance-sampling ratios. However, when bootstrapping is used, larger trajectories are given smaller weights, and hence their corresponding importance-sampling ratios will have a less severe effect on the updates. For example, when full bootstrapping is used, that is, $\lambda_k = 0, \forall k$, the overall return becomes $G_{k,t+1}^{\rho} = \rho_k \left(R_{k+1} + \gamma_{k+1} \boldsymbol{\phi}_{k+1}^\top \boldsymbol{\phi}_k \right)$, with an importance-sampling ratio of a one-transition long trajectory. In such cases, updating $\mathbf{u}$ with the importance-sampling ratio of the full trajectory ρ_k^{t+1} is unnecessary. Hence, the amount of importance weighting in $\mathbf{u}$ at each step should be modulated by the amount of discounting and bootstrapping.

Based on the overall return in (7.17) and the idea of discounting and bootstrapping-*aware* update of $\mathbf{u}$ discussed above, we propose a new off-policy TD algorithm based on WIS, which we call *WIS-TD*(λ). It consists of the following forward-view updates:

$$\tilde{\rho}_k^{t+1} \stackrel{\text{def}}{=\!=} \rho_k \sum_{i=k+1}^{t} C_k^{i-1} (1 - \gamma_i \lambda_i) + \rho_k C_k^t; \quad \tilde{\rho}_t^t \stackrel{\text{def}}{=\!=} 0, \tag{7.18}$$

$$\mathbf{u}_{k+1}^{t+1} \stackrel{\text{def}}{=\!=} (\mathbf{1} - \eta \boldsymbol{\phi}_k \circ \boldsymbol{\phi}_k) \circ \mathbf{u}_k^{t+1} + \tilde{\rho}_k^{t+1} \boldsymbol{\phi}_k \circ \boldsymbol{\phi}_k, \tag{7.19}$$

$$\boldsymbol{\alpha}_{k+1} \stackrel{\text{def}}{=\!=} \mathbf{1} \oslash \mathbf{u}_{k+1}^{k+1}, \tag{7.20}$$

$$\boldsymbol{\theta}_{k+1}^{t+1} \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_{k}^{t+1} + \boldsymbol{\alpha}_{k+1} \circ \left(G_{k,t+1}^{\rho} - \rho_k \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^{t+1}\right) \boldsymbol{\phi}_k. \tag{7.21}$$

It can be easily verified that, when no bootstrapping is used, that is, $\lambda_k = 1, \forall k$ and discounting occurs only at the data horizon $t+1$, that is, $\gamma_0 = \gamma_1 = \cdots = \gamma_t = 1$ and $\gamma_{t+1} = 0$, then $\tilde{\rho}_k^{t+1} = \rho_k^{t+1}$, and we already showed that the target of WIS-TD(λ) $G_{k,t+1}^{\rho}$ reduces to the WIS-SGD-2 target in this case. Hence, WIS-TD(λ) subsumes WIS-SGD-2, establishing a direct backward consistency to sample average.

In the following, we apply the new general equivalence technique, we provided in Theorem 25, to the above forward-view update to derive an $O(n)$ backward-view update computing the same parameter vector $\boldsymbol{\theta}_t$ at each t. For that, first we derive an $O(n)$ backward-view update for the step size that computes the same $\boldsymbol{\alpha}_t$ as in the above algorithm at each t.

Theorem 29 (Backward view update for $\boldsymbol{\alpha}_t$ of WIS-TD(λ))**.** *The step-size vector $\boldsymbol{\alpha}_t$ computed by the following backward-view update and the forward-view update defined by (7.18) – (7.20) are equal at each step t:*

$$\mathbf{u}_{t+1} \stackrel{\text{def}}{=\!=} (\mathbf{1} - \eta \boldsymbol{\phi}_t \circ \boldsymbol{\phi}_t) \circ \mathbf{u}_t + \rho_t \boldsymbol{\phi}_t \circ \boldsymbol{\phi}_t + (\rho_t - 1)\gamma_t \lambda_t (\mathbf{1} - \eta \boldsymbol{\phi}_t \circ \boldsymbol{\phi}_t) \circ \mathbf{v}_t, \tag{7.22}$$

$$\mathbf{v}_{t+1} \stackrel{\text{def}}{=\!=} \gamma_t \lambda_t \rho_t (\mathbf{1} - \eta \boldsymbol{\phi}_t \circ \boldsymbol{\phi}_t) \circ \mathbf{v}_t + \rho_t \boldsymbol{\phi}_t \circ \boldsymbol{\phi}_t, \tag{7.23}$$

$$\boldsymbol{\alpha}_{t+1} \stackrel{\text{def}}{=\!=} \mathbf{1} \oslash \mathbf{u}_{t+1}. \tag{7.24}$$

Proof. First, note that the component-wise vector multiplication in (7.19) can be written equivalently as a matrix-vector multiplication in the following way:

$$(\mathbf{1} - \eta \boldsymbol{\phi}_k \circ \boldsymbol{\phi}_k) \circ \mathbf{u}_k^{t+1} = (\mathbf{I} - \eta \text{Diag}(\boldsymbol{\phi}_k \circ \boldsymbol{\phi}_k)) \mathbf{u}_k^{t+1},$$

where $\text{Diag}(\mathbf{v}) \in \mathbb{R}^{|\mathbf{v}| \times |\mathbf{v}|}$ is a diagonal matrix with the components of $\mathbf{v}$ in its diagonal.

In Theorem 25, we substitute $\boldsymbol{\theta}_k^{t+1} = \mathbf{u}_k^{t+1}$, $\mathbf{F}_k = (\mathbf{I} - \eta \text{Diag}(\boldsymbol{\phi}_k \circ \boldsymbol{\phi}_k))$, $\mathbf{x}_k = \mathbf{0}$, $\mathbf{w}_k = \boldsymbol{\phi}_k \circ \boldsymbol{\phi}_k$ and $Y_k^{t+1} = \tilde{\rho}_k^{t+1}$.

Now, $\tilde{\rho}_k^{t+1}$ can be recursively in t written as follows

$$\begin{aligned}
\tilde{\rho}_k^{t+1} &= \rho_k \sum_{i=k+1}^{t} C_k^{i-1}(1 - \gamma_i \lambda_i) + \rho_k C_k^t \\
&= \rho_k \sum_{i=k+1}^{t-1} C_k^{i-1}(1 - \gamma_i \lambda_i) + \rho_k C_k^{t-1}(1 - \gamma_t \lambda_t) + \rho_k C_k^t \\
&= \rho_k \sum_{i=k+1}^{t-1} C_k^{i-1}(1 - \gamma_i \lambda_i) + \rho_k C_k^{t-1} + \rho_k C_k^{t-1} \rho_t \gamma_t \lambda_t - \rho_k C_k^{t-1} \gamma_t \lambda_t \\
&= \tilde{\rho}_k^t + (\rho_t - 1)\gamma_t \lambda_t \rho_k C_k^{t-1}.
\end{aligned}$$

Hence, it proves that

$$Y_k^{t+1} - Y_k^t = d_{k+1}\left(Y_{k+1}^{t+1} - Y_{k+1}^t\right) + b_t g_k \prod_{j=k+1}^{t-1} c_j, 0 \leq k < t,$$

with $d_i = 0$, $b_i = (\rho_i - 1)\gamma_i\lambda_i$, $g_i = \rho_i$ and $c_i = \gamma_i\lambda_i\rho_i, \forall i$.

Inserting these substitutes in Theorem 25 yields us the backward-view defined by (7.22) – (7.24). □

Now, we derive an $O(n)$ backward-view update that computes the same $\boldsymbol{\theta}_t^t$ as the above forward view.

Theorem 30 (Backward view update for $\boldsymbol{\theta}_t^t$ of WIS-TD(λ))**.** *The parameter vector $\boldsymbol{\theta}_t$ computed by the following backward-view update and the parameter vector $\boldsymbol{\theta}_t^t$ computed by the forward-view update defined by (7.17) and (7.21) are equal at every time step t:*

$$\begin{aligned}
\mathbf{e}_t &\stackrel{\text{def}}{=\!=} \rho_t\boldsymbol{\alpha}_{t+1} \circ \boldsymbol{\phi}_t \\
&\quad + \gamma_t\lambda_t\rho_t \left(\mathbf{e}_{t-1} - \rho_t \left(\boldsymbol{\alpha}_{t+1} \circ \boldsymbol{\phi}_t\right) \boldsymbol{\phi}_t^\top \mathbf{e}_{t-1}\right), && (7.25)\\
\boldsymbol{\theta}_{t+1} &\stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_t + \boldsymbol{\alpha}_{t+1} \circ \rho_t \left(\boldsymbol{\theta}_{t-1}^\top\boldsymbol{\phi}_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) \boldsymbol{\phi}_t \\
&\quad + (R_{t+1} + \gamma_{t+1}\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_{t-1}^\top\boldsymbol{\phi}_t)\mathbf{e}_t \\
&\quad + (\rho_t - 1)\gamma_t\lambda_t \left(\mathbf{d}_t - \rho_t \left(\boldsymbol{\alpha}_{t+1} \circ \boldsymbol{\phi}_t\right) \boldsymbol{\phi}_t^\top \mathbf{d}_t\right), && (7.26)\\
\mathbf{d}_{t+1} &\stackrel{\text{def}}{=\!=} \gamma_t\lambda_t\rho_t \left(\mathbf{d}_t - \rho_t \left(\boldsymbol{\alpha}_{t+1} \circ \boldsymbol{\phi}_t\right) \boldsymbol{\phi}_t^\top \mathbf{d}_t\right) \\
&\quad + \left(R_{t+1} + \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_{t-1}^\top\boldsymbol{\phi}_t\right) \mathbf{e}_t. && (7.27)
\end{aligned}$$

Proof. First, we redefine (7.21) for convenience:

$$\boldsymbol{\theta}_{k+1}^{t+1} \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_k^{t+1} + \boldsymbol{\alpha}_{k+1} \circ \rho_k \left(\zeta_{k,t+1}^\rho - \boldsymbol{\phi}_k^\top \boldsymbol{\theta}_k^{t+1}\right) \boldsymbol{\phi}_k, \tag{7.28}$$

where $G_{k,t+1}^\rho = \rho_k\zeta_{k,t+1}^\rho$. Hence, $\zeta_{k,t+1}^\rho$ can be given by:

$$\begin{aligned}
\zeta_{k,t+1}^\rho \stackrel{\text{def}}{=\!=} C_k^t \Big((1 - \gamma_{t+1})G_k^{t+1} + \gamma_{t+1} \left(G_k^{t+1} + \boldsymbol{\phi}_{t+1}^\top\boldsymbol{\theta}_t\right)\Big) \\
+ \sum_{i=k+1}^{t} C_k^{i-1} \Big((1 - \gamma_i)G_k^i + \gamma_i(1 - \lambda_i) \left(G_k^i + \boldsymbol{\phi}_i^\top \boldsymbol{\theta}_{i-1}\right)\Big) \\
- \left(C_k^t + \sum_{i=k+1}^{t} C_k^{i-1}(1 - \gamma_i\lambda_i) - 1\right) \boldsymbol{\phi}_k^\top\boldsymbol{\theta}_{k-1}.
\end{aligned}$$

In Theorem 25, we substitute $\mathbf{F}_k = \mathbf{I} - \rho_k(\boldsymbol{\alpha}_{k+1} \circ \boldsymbol{\phi}_k)\boldsymbol{\phi}_k^\top$, $\mathbf{w}_k = \rho_k\boldsymbol{\alpha}_{k+1} \circ \boldsymbol{\phi}_k$, $Y_k^{t+1} = \zeta_{k,t+1}^\rho$ and $\mathbf{x}_k = 0, \forall k$, to get (7.28). Now, the next step is to establish a recursive relation for ζ^ρ both in k and t. For that, we use the following identities:

$$\begin{aligned}
G_k^{k+1} &= R_{k+1}, \\
G_k^{t+1} &= \sum_{i=k}^{t} R_{i+1} = R_{k+1} + G_{k+1}^{t+1}.
\end{aligned}$$

First we establish the recurrence relation in k:

$$
\begin{aligned}
\zeta^{\rho}_{k,t+1} &= C^t_k\Big((1-\gamma_{t+1})G^{t+1}_k + \gamma_{t+1}\left(G^{t+1}_k + \boldsymbol{\phi}^\top_{t+1}\boldsymbol{\theta}_t\right)\Big) \\
&\quad + \sum_{i=k+1}^{t} C^{i-1}_k\Big((1-\gamma_i)G^i_k + \gamma_i(1-\lambda_i)\left(G^i_k + \boldsymbol{\phi}^\top_i\boldsymbol{\theta}_{i-1}\right)\Big) \\
&\quad - \left(C^t_k + \sum_{i=k+1}^{t} C^{i-1}_k(1-\gamma_i\lambda_i) - 1\right)\boldsymbol{\phi}^\top_k\boldsymbol{\theta}_{k-1} \\
&= C^t_k\Big((1-\gamma_{t+1})\left(R_{k+1} + G^{t+1}_{k+1}\right) + \gamma_{t+1}\left(R_{k+1} + G^{t+1}_{k+1} + \boldsymbol{\phi}^\top_{t+1}\boldsymbol{\theta}_t\right)\Big) \\
&\quad + \Big((1-\gamma_{k+1})G^{k+1}_k + \gamma_{k+1}(1-\lambda_{k+1})\left(G^{k+1}_k + \boldsymbol{\phi}^\top_{k+1}\boldsymbol{\theta}_k\right)\Big) \\
&\quad + \sum_{i=k+2}^{t} C^{i-1}_k\Big((1-\gamma_i)\left(R_{k+1} + G^i_{k+1}\right) + \gamma_i(1-\lambda_i)\left(R_{k+1} + G^i_{k+1} + \boldsymbol{\phi}^\top_i\boldsymbol{\theta}_{i-1}\right)\Big) \\
&\quad - \left(C^t_k + \sum_{i=k+1}^{t} C^{i-1}_k(1-\gamma_i\lambda_i) - 1\right)\boldsymbol{\phi}^\top_k\boldsymbol{\theta}_{k-1} \\
&= \rho_{k+1}\gamma_{k+1}\lambda_{k+1}C^t_{k+1}\Big((1-\gamma_{t+1})G^{t+1}_{k+1} + \gamma_{t+1}(G^{t+1}_{k+1} + \boldsymbol{\phi}^\top_{t+1}\boldsymbol{\theta}_t)\Big) \\
&\quad + \rho_{k+1}\gamma_{k+1}\lambda_{k+1}\sum_{i=k+2}^{t} C^{i-1}_{k+1}\Big((1-\gamma_i)G^i_{k+1} + \gamma_i(1-\lambda_i)\left(G^i_{k+1} + \boldsymbol{\phi}^\top_i\boldsymbol{\theta}_{i-1}\right)\Big) \\
&\quad - \rho_{k+1}\gamma_{k+1}\lambda_{k+1}\left(C^t_{k+1} + \sum_{i=k+2}^{t} C^{i-1}_{k+1}(1-\gamma_i\lambda_i) - 1\right)\boldsymbol{\phi}^\top_{k+1}\boldsymbol{\theta}_k \\
&\quad + \left(C^t_k + \sum_{i=k+2}^{t} C^{i-1}_k(1-\gamma_i\lambda_i) - \rho_{k+1}\gamma_{k+1}\lambda_{k+1}\right)\boldsymbol{\phi}^\top_{k+1}\boldsymbol{\theta}_k \\
&\quad + C^t_k R_{k+1} + (1-\gamma_{k+1}\lambda_{k+1})R_{k+1} + \gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}^\top_{k+1}\boldsymbol{\theta}_k \\
&\quad + R_{k+1}\sum_{i=k+2}^{t} C^{i-1}_k(1-\gamma_i\lambda_i) \\
&\quad - \left(C^t_k + \sum_{i=k+1}^{t} C^{i-1}_k(1-\gamma_i\lambda_i) - 1\right)\boldsymbol{\phi}^\top_k\boldsymbol{\theta}_{k-1} \\
&= \rho_{k+1}\gamma_{k+1}\lambda_{k+1}\zeta^{\rho}_{k+1,t+1} \\
&\quad + \left(C^t_k + \sum_{i=k+1}^{t} C^{i-1}_k(1-\gamma_i\lambda_i) - 1\right)\left(R_{k+1} + \boldsymbol{\phi}^\top_{k+1}\boldsymbol{\theta}_k - \boldsymbol{\phi}^\top_k\boldsymbol{\theta}_{k-1}\right) \\
&\quad + R_{k+1} + \boldsymbol{\phi}^\top_{k+1}\boldsymbol{\theta}_k - \rho_{k+1}\gamma_{k+1}\lambda_{k+1}\boldsymbol{\phi}^\top_{k+1}\boldsymbol{\theta}_k \\
&\quad + \gamma_{k+1}(1-\lambda_{k+1})\boldsymbol{\phi}^\top_{k+1}\boldsymbol{\theta}_k - (1-\gamma_{k+1}\lambda_{k+1})\boldsymbol{\phi}^\top_{k+1}\boldsymbol{\theta}_k \\
&= \rho_{k+1}\gamma_{k+1}\lambda_{k+1}\zeta^{\rho}_{k+1,t+1} \\
&\quad + \left(C^t_k + \sum_{i=k+1}^{t} C^{i-1}_k(1-\gamma_i\lambda_i) - 1\right)\left(R_{k+1} + \boldsymbol{\phi}^\top_{k+1}\boldsymbol{\theta}_k - \boldsymbol{\phi}^\top_k\boldsymbol{\theta}_{k-1}\right) \\
&\quad + R_{k+1} + \gamma_{k+1}\left(1-\rho_{k+1}\lambda_{k+1}\right)\boldsymbol{\phi}^\top_{k+1}\boldsymbol{\theta}_k.
\end{aligned}
$$

Then the recurrence in t can be established by subtracting $\zeta^{\rho}_{k,t}$ from $\zeta^{\rho}_{k,t+1}$:

$$
\begin{aligned}
\zeta^{\rho}_{k,t+1} - \zeta^{\rho}_{k,t} &\stackrel{\text{def}}{=\!=} \rho_{k+1}\gamma_{k+1}\lambda_{k+1}\zeta^{\rho}_{k+1,t+1} \\
&\quad + \left(C^{t}_{k} + \sum_{i=k+1}^{t} C^{i-1}_{k}(1-\gamma_i\lambda_i) - 1\right)\left(R_{k+1} + \boldsymbol{\phi}^{\top}_{k+1}\boldsymbol{\theta}_k - \boldsymbol{\phi}^{\top}_{k}\boldsymbol{\theta}_{k-1}\right) \\
&\quad + R_{k+1} + \gamma_{k+1}\left(1-\rho_{k+1}\lambda_{k+1}\right)\boldsymbol{\phi}^{\top}_{k+1}\boldsymbol{\theta}_k - \rho_{k+1}\gamma_{k+1}\lambda_{k+1}\zeta^{\rho}_{k+1,t} \\
&\quad - \left(C^{t-1}_{k} + \sum_{i=k+1}^{t-1} C^{i-1}_{k}(1-\gamma_i\lambda_i) - 1\right)\left(R_{k+1} + \boldsymbol{\phi}^{\top}_{k+1}\boldsymbol{\theta}_k - \boldsymbol{\phi}^{\top}_{k}\boldsymbol{\theta}_{k-1}\right) \\
&\quad - R_{k+1} + \gamma_{k+1}\left(1-\rho_{k+1}\lambda_{k+1}\right)\boldsymbol{\phi}^{\top}_{k+1}\boldsymbol{\theta}_k \\
&= \rho_{k+1}\gamma_{k+1}\lambda_{k+1}\left(\zeta^{\rho}_{k+1,t+1} - \zeta^{\rho}_{k+1,t}\right) \\
&\quad + \left(C^{t}_{k} - C^{t-1}_{k} + C^{t-1}_{k}(1-\gamma_t\lambda_t)\right)\left(R_{k+1} + \boldsymbol{\phi}^{\top}_{k+1}\boldsymbol{\theta}_k - \boldsymbol{\phi}^{\top}_{k}\boldsymbol{\theta}_{k-1}\right) \\
&= \rho_{k+1}\gamma_{k+1}\lambda_{k+1}\left(\zeta^{\rho}_{k+1,t+1} - \zeta^{\rho}_{k+1,t}\right) \\
&\quad + (\rho_t - 1)\gamma_t\lambda_t C^{t-1}_{k}\left(R_{k+1} + \boldsymbol{\phi}^{\top}_{k+1}\boldsymbol{\theta}_k - \boldsymbol{\phi}^{\top}_{k}\boldsymbol{\theta}_{k-1}\right).
\end{aligned}
$$

The above recurrence relation establishes

$$
Y^{t+1}_{k} - Y^{t}_{k} = d_{k+1}\left(Y^{t+1}_{k+1} - Y^{t}_{k+1}\right) + b_t g_k \prod_{j=k+1}^{t-1} c_j, 0 \le k < t,
$$

with $d_i = \rho_i\gamma_i\lambda_i$, $b_i = (\rho_i - 1)\gamma_i\lambda_i$, $g_i = R_{i+1} + \boldsymbol{\phi}^{\top}_{i+1}\boldsymbol{\theta}_i - \boldsymbol{\phi}^{\top}_{i}\boldsymbol{\theta}_{i-1}$ and $c_i = \gamma_i\lambda_i\rho_i, \forall i$. Inserting these substitutes in Theorem 25 yields us the backward-view defined by (7.25) – (7.27) □

The overall backward view of WIS-TD(λ) is defined by (7.22) – (7.27). Note that Theorem 30 does not depend on how $\boldsymbol{\alpha}_{t+1}$ is set. The per-update time and memory complexity of WIS-TD(λ) is $O(n)$. An auxiliary parameter vector might be included in WIS-TD(λ) by making use of the $\mathbf{x}_k$ vector of the online equivalence technique as was done by van Hasselt et al. (2014), but we do not explore this possibility here.

The vector step-size adaptation based on the update of the usage vector $\mathbf{u}$ is only loosely coupled with WIS-TD(λ) and can be freely combined with existing off-policy algorithms as well as the on-policy ones. When combined with the existing algorithms, this step-size adaptation is expected to yield benefits due to the rescaling it performs according to the magnitude of importance-sampling weights and the frequency of feature activation.

We propose two new off-policy algorithms: *WIS-GTD*(λ) and *WIS-TO-GTD*(λ), based on GTD(λ) (Maei 2011) and true online GTD(λ) (van Hasselt et al. 2014), respectively. In both algorithms, we propose replacing the scalar step size of the main parameter vector with the vector step size according to (7.22) – (7.24). The scalar step-size parameter of the auxiliary parameter vector of GTD(λ) and true online GTD(λ) could also be replaced with the vector step size with a different recency-weighting factor, but we leave it out here.

We propose two new on-policy algorithms: *usage-based TD*(λ) (U-TD(λ)) and *usage-based true online TD*(λ) (U-TO-TD(λ)), by combining the vector-step-size adaptation with

two existing on-policy algorithms: TD(λ) (Sutton & Barto 1998) and true online TD(λ) (van Seijen & Sutton 2014), respectively. There are interesting interrelationships between these on-policy and off-policy algorithms. For example, WIS-GTD(λ) becomes equivalent to U-TD(λ) in the on-policy case when the second step-size parameter $\beta = 0$. On the other hand, WIS-TD(λ) directly degenerates to U-TO-TD(λ) in the on-policy case, whereas WIS-TO-GTD(λ) reduces to U-TO-TD(λ) in the on-policy case with $\beta = 0$.

7.4 Experimental Results

In this section we evaluate the new algorithms using two sets of experiments with off-policy and on-policy policy-evaluation tasks, respectively. Source code for both the off-policy and on-policy experiments are available online In the first set of experiments, we compared the new off-policy algorithms: WIS-TD(λ), WIS-GTD(λ) and WIS-TO-GTD(λ) with two existing $O(n)$ algorithms: GTD(λ) and true online GTD(λ) (TO-GTD(λ)), and with two least squares algorithms: LSTD-TO(λ), an off-policy algorithm proposed by Dann, Neumann and Peters (2014), and WIS-LSTD(λ), an ideal extension of WIS. For evaluation, we created three off-policy policy-evaluation tasks.

The first task was constructed based on a random-walk Markov chain where the states can be imagined to be laid out on a horizontal line. There were 11 non-terminal states and two terminal states: on the left and the right ends of the chain. From each non-terminal state, there were two actions available: `left`, leads to the state to the left, and `right`, leads to the state to the right. The initial state was always set to the state in the middle of the chain. The reward was sparse: 0 for all transitions except for the rightmost transition to the terminal state, where it was +1. The behavior policy was uniformly random between the two actions and the target policy chose `right` with 0.99 probability. No discounting was used. The feature vectors were binary representations of state indices. For 11 non-terminal states, each feature vector was of length $\lfloor \log_2(11) \rfloor + 1 = 4$, and these vectors for the states from left to right were $(0,0,0,1)^\top$, $(0,0,1,0)^\top$, $(0,0,1,1,)^\top, \cdots, (1,0,1,1)^\top$. The features were all zero for the terminal states.

The second and the third tasks were constructed using randomly generated MDPs. We represent a randomly generated MDP as $(N, M, b, \mathbf{\Gamma})$ where N and M stand for the number of states and actions, respectively, and b is a branching factor denoting the number of next states for a given state-action pair. Here, $\mathbf{\Gamma} \in \mathbb{R}^{N \times N}$ is a diagonal matrix where the entries are the state-dependent discounting $\gamma(\cdot)$ for each state. We use such a state-dependent discounting to denote termination under the target policy while experience continues seamlessly under the behavior policy. For each state, the next b states were chosen from total N states randomly without replacement, and the transition probabilities were generated by partitioning the unit interval at $b - 1$ cut points which were selected uniformly randomly from $[0, 1]$. The rewards for a transition from a state-action pair to the next state were selected uniformly randomly from $[0, 1]$ and kept deterministic. The behavior policy

probabilities for different actions in a particular state were set using uniform random numbers from $[10^{-15}, 1 + 10^{-15}]$ and normalized to sum to one. The target policy is much less stochastic: one of the actions in a particular state is chosen to have probability 0.99 and the rest of the actions are equiprobable.

We constructed the second task by randomly generating an MDP with parameters (10, $3, 3, \mathbf{\Gamma}$), where $\gamma(\cdot) = 0$ for 2 randomly chosen states to denote termination under the target policy and $\gamma(\cdot) = 0.99$ for the rest of the 8 states. For the third task, we randomly generated an MDP with parameters $(100, 3, 10, \mathbf{\Gamma})$, where $\gamma(\cdot) = 0$ for 5 randomly chosen states and $\gamma(\cdot) = 0.99$ for the rest. The feature vectors were binary representations of the indices of all states including those for which $\gamma(\cdot) = 0$. In these two tasks, the feature vectors were normalized to have unit length.

We tested all algorithms for different values of constant λ, from 0 to 0.9 in steps of 0.1 and from 0.9 to 1.0 in steps of 0.01. The first step-size parameter α of GTD(λ) and TO-GTD(λ) was varied by powers of 10 with powers chosen from -3 to 0 in steps of 0.25. The second step-size parameter β of both algorithms was varied among values $[0, 0.001, 0.01, 0.1]$. The initial value u_0 of the components of the usage vector $\mathbf{u}$ for WIS-TD(λ), WIS-GTD(λ) and WIS-TO-GTD(λ) was varied by powers of 10 with powers chosen from 0 to 3 in steps of 0.25. The recency-weighting factor η of the same algorithms was set as $\eta = \mu/u_0$, where μ was varied among values $[0, 0.001, 0.01, 0.1, 1]$. The second step-size parameter β for WIS-GTD(λ) and WIS-TO-GTD(λ) was set to zero. The matrix to be inverted in LSTD-TO(λ) and WIS-LSTD(λ) was initialized to $\epsilon\mathbf{I}$, where ϵ was varied by powers of 10 with powers chosen from -3 to $+3$ in steps of 0.2. The initial parameter vector $\boldsymbol{\theta}_0$ was set to $\mathbf{0}$.

Performance was measured as the empirical mean squared error (MSE) between the estimated values of the states and their true values under the target policy projected to the space spanned by the given features. The error was weighted according to the state-visitation distribution under the behavior policy. As the scale of this MSE measure can vary between these tasks, we normalized it by the squared weighted L2 norm of the projected true value, which is equivalent to the MSE under $\boldsymbol{\theta} = \mathbf{0}$. As a result, the initial normalized MSE (NMSE) for each algorithm was 1. For each run, we averaged this error over 100 episodes measured at the end of each episode for the first task, over 500 steps for the second task, and over 5000 steps for the third. We produced the final estimate by further averaging over 50 independent runs.

Figure 7.1 shows the empirical performance together with the standard error on the three off-policy policy-evaluation tasks with respect to different λ and optimized over all other parameters. In all three tasks, the new algorithms significantly outperformed both GTD(λ) and TO-GTD(λ) indicating the effectiveness of the adaptive vector step size in retaining the advantage of WIS. The new algorithms also performed competitively with LSTD-TO(λ) in all tasks. Among the new algorithms, WIS-GTD(λ) had superior performance with large

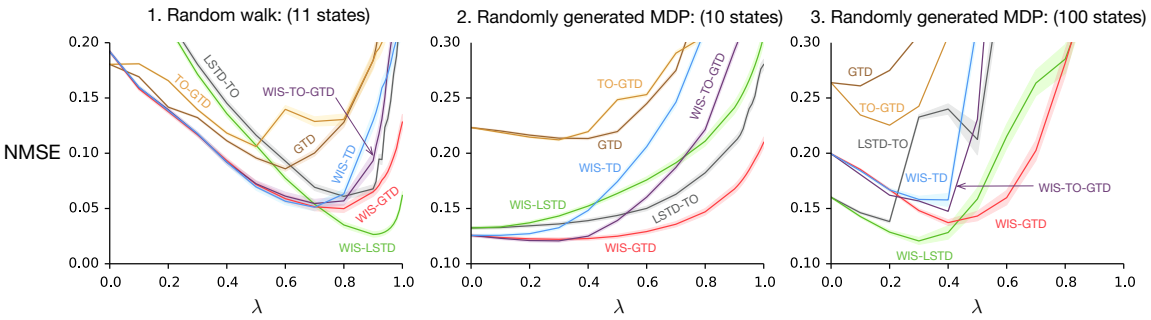

Figure 7.1: Empirical comparison of the new WIS-based $O(n)$ algorithms with two existing $O(n)$ algorithms and two LSTD algorithms on three off-policy policy-evaluation tasks. Performance is shown in the empirical normalized MSE (NMSE) measured by averaging over 50 independent runs and 100 episodes for the first task, 500 steps for the second, and 5000 steps for the third. The new WIS-based algorithms performed significantly better than both existing $O(n)$ algorithms in all three off-policy tasks and competitively with one of the LSTD algorithms.

values of λ.

We also studied the sensitivity of the new algorithms with respect to their parameters. Although these algorithms replace the scalar constant step-size parameter of their base learner with an adaptive vector step size based on feature usage, the estimate of the usage depends on two new parameters: the initial value u_0 and the recency weighting constant η. The initial value u_0 of the usage vector can be interpreted as the inverse of the initial step size, and its tuning can be as extensive as that of the scalar step-size parameter in other algorithms. On the other hand, η can be viewed as the desired final step size. As a result, their product $\mu = u_0\eta$ is unit free and requires less rigorous tuning.

In our final set of experiments, we compared the new $O(n)$ on-policy algorithms: U-TD(λ) and U-TO-TD(λ), with two $O(n)$ on-policy algorithms: TD(λ) with accumulating traces and true online TD(λ), which we call *TO-TD*(λ).

We used randomly generated MDPs to produce two on-policy policy-evaluation tasks. As the TD algorithms here estimate state-value functions, it sufficed to construct Markov Reward Processes (MRPs), which we obtained by choosing the number of actions M to be 1 in both tasks. Our first task used an MDP with 10 states: $(10, 1, 3, 0.99\mathbf{I})$ and the second task used an MDP with 100 states: $(100, 1, 10, 0.99\mathbf{I})$. The feature vectors were binary representations of the state indices as in the off-policy tasks and were normalized to have unit length.

For each task, the performance of each algorithm was measured for different parameter values. For TD(λ) and TO-TD(λ), the scalar step-size parameter α was varied by powers of 10 with powers chosen from -3 to 1 in steps of 0.25. For U-TD(λ) and U-TO-TD(λ), the parameter u_0 was varied by powers of 10 with powers chosen from -1 to 3 in steps of 0.25. The rest of the parameters for all four algorithms were varied using the same values

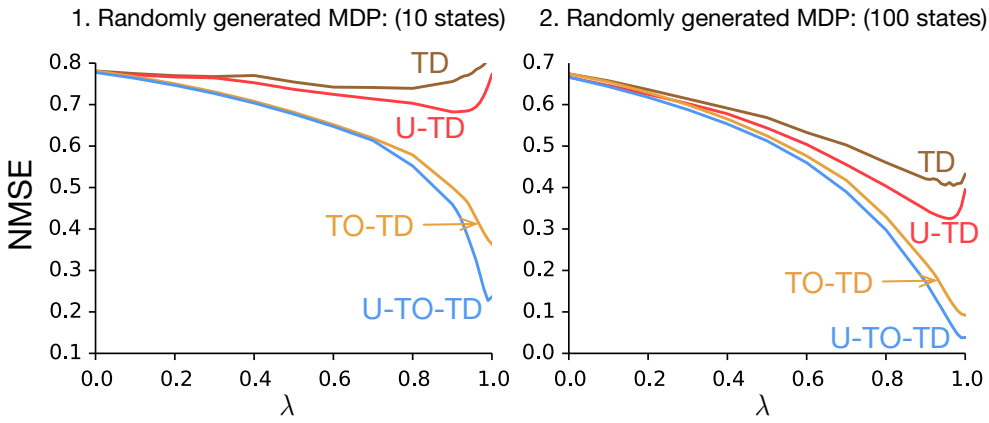

Figure 7.2: Empirical comparison of the new $O(n)$ usage-based algorithms with two existing $O(n)$ TD algorithms on on-policy policy-evaluation tasks. Performance is measured in empirical normalized MSE (NMSE).

as in the off-policy tasks. Performance was measured using NMSE as in the off-policy tasks. For each run, we averaged this error over 100 steps for the first task and 1000 steps for the second. The final estimate is produced again by averaging over 50 independent runs.

Figure 7.2 shows the performance on both tasks for different λ with the rest of the parameters optimized. Left plot corresponds to MDP $(10, 1, 3, 0.99\mathbf{I})$ and the right plot corresponds to MDP $(100, 1, 10, 0.99\mathbf{I})$. On both tasks, the new algorithms performed significantly better than their base learning algorithms for higher values of λ and performed equally well for the smaller ones. The standard error in each case was smaller than the width of the curves shown. This set of experiments suggests that the step-size adaptation based on the usage of features can be useful in both off-policy and on-policy tasks.

7.5 Discussion and Conclusions

In this chapter, we carried over much of the benefits of WIS to $O(n)$ off-policy algorithms. In the process, we developed a modification of stochastic gradient descent that are more closely related to sample averages. The key idea behind this modification is to maintain a "usage" vector to keep track how much each feature is used in the updates and set the step-size parameter inversely proportional to that measure. We developed new $O(n)$ off-policy algorithms that incorporated this modification. On three off-policy policy-evaluation experiments, the new algorithms outperformed the existing $O(n)$ off-policy algorithms and performed competitively with LSTD-TO(λ). However, none of these algorithms are backward compatible to tabular WIS estimators. In our experiments, the off-policy algorithms with large values of λ did not perform well compared to the best performance they achieved. This held for both SGD-based and LS-based algorithms including the existing and the new algorithms. This shows that multi-step learning is difficult to utilize in off-policy algorithms.

Chapter 8

Multi-step Off-policy Learning without Importance Sampling Ratios [1]

This chapter entirely contains our contributions toward the issue of high variance using the technique of bootstrapping, an approach different than weighted importance sampling. These contributions comprise a new off-policy algorithm with reduced estimation variance and linear computational complexity and a new framework for analyzing some of the existing off-policy algorithms.

The main source of variance in importance sampling is the importance sampling ratio. Weighted importance sampling (WIS) addressed this issue by bounding its net effect. However, importance sampling is still present in WIS. Its extension to SGD with linear computational complexity WIS-TD(λ) does not fully carry over the benefits of WIS. In general, we have observed in the experiments of the previous chapter that off-policy algorithms do not utilize multi-step learning very well; their performance deteriorates when a large value of λ is chosen. It is because the large variance issue with importance sampling is the most severe in multi-step learning (White 2015, Mahmood & Sutton 2015). Consequently, multi-step off-policy learning remains problematic and largely unfulfilled.

An obvious approach to solve the problem of multi-step off-policy learning would then be to develop an algorithm that avoids using importance-sampling ratios. The absence of these ratios will presumably reduce the estimation variance, making long-term multi-step learning tenable. Only a few model-free algorithms have been proposed to learn off-policy without using importance-sampling ratios (Precup et al. 2000, van Hasselt 2011, Harutyunyan et al. 2016). However, all these algorithms were introduced either for one-step learning (van Hasselt 2011) or for learning with lookup table representation (Precup et al. 2000, Harutyunyan et al. 2016). Multi-step learning does not have a lasting influence on performance in this case.

Our key contribution in this chapter is to develop an algorithmic technique based on

[1] This chapter is adapted from a paper coauthored by this author (Mahmood, Yu & Sutton 2017).

modulating the amount to which the estimates of the subsequent states are used, a concept known as *bootstrapping*, in an action-dependent manner. It results in an action-dependent bootstrapping parameter, which is a generalization of the state-dependent bootstrapping parameter used in prior works (Maei & Sutton 2010, Sutton et al. 2014). For action-value estimation, we show that importance-sampling ratios can be eliminated by varying the action-dependent bootstrapping parameter for different state-action pairs in a particular way. We introduce a new algorithm called *ABQ* using this technique that can achieve much less estimation variance compared to the state-of-the-art off-policy algorithm. ABQ is the first to effectively achieve multi-step function approximation solutions for off-policy learning without explicitly using importance-sampling ratios. However, it is possible to produce other off-policy algorithms with bounded estimation variance by setting the action-dependent bootstrapping parameter in other ways, giving rise to the *action-dependent bootstrapping framework*. This allows analyzing some of the existing off-policy algorithms as well as derive new ones. A prior algorithm, Tree Backup (Precup et al. 2000), can be retrieved as a special case of our algorithm. Furthermore, we show that another off-policy algorithm, Retrace (Munos et al. 2016), can also be derived under the action-dependent bootstrapping framework. Our analysis allows an extension of Retrace to the case of function approximation with stability, giving rise to another algorithm based on the action-dependent bootstrapping technique, we call *AB-Trace*.

8.1 Formulation of the Action-value Estimation Task

In this section, we formulate the off-policy learning task with parametric function approximation for action-value function. So far, we have developed learning algorithms for state-value estimation. The problem formulation for action-value estimation is not much different from that of state-value estimation we described in Chapter 2. We simplify the problem formulation here by considering a state-independent constant discount factor $\gamma < 1$, which is a special case of in the GVF framework and common in continuing prediction tasks. Derivations of algorithms are simpler in this setting, but the resulting algorithms can be easily extended back in the GVF framework. In that sense, this setting does not diminish the contribution we provide here.

As before, we consider an agent in a dynamical environment with a finite state space $\mathcal{S}$ and action space $\mathcal{A}$. At each time $t = 0, 1, \ldots$, if the present state is $s \in \mathcal{S}$ and the agent takes action $a \in \mathcal{A}$, the next state S_{t+1} is $s' \in \mathcal{S}$ with probability $p(s'|s, a)$, and the agent receives a random reward R_{t+1} with mean $r(s, a)$ and finite variance upon the state transition. A randomized stationary policy π specifies the probability $\pi(a|s)$ of taking action a at state s. Of our interest is a given policy π, referred to as the target policy, and the performance of the agent if it follows π. Specifically, our interest in this paper is to estimate the action-value function of π, defined as the expected sum of discounted rewards

for any initial state-action pair (s, a):

$$q_\pi(s,a) \stackrel{\text{def}}{=\!=} \mathrm{E}_\pi\left[\sum_{t=1}^{\infty}\gamma^{t-1}R_t\Big|S_0=s, A_0=a\right]. \tag{8.1}$$

Important to multi-step learning are multi-step Bellman equations satisfied by the action-value function q_π. We review here such equations for the well-known TD(λ), where $\lambda \in [0,1]$ is the bootstrapping parameter. Let $\mathbf{P}_\pi$ be the transition probability matrix of the Markov chain on $\mathcal{S}\times\mathcal{A}$ induced by the target policy π, and let $\mathbf{r}\in\mathbb{R}^{|\mathcal{S}|\cdot|\mathcal{A}|}$ be the vector of expected rewards for different state-action pairs: $[\mathbf{r}]_{sa} \stackrel{\text{def}}{=\!=} r(s,a)$. For $\lambda\in[0,1]$, define the multi-step Bellman operator $T_\pi^{(\lambda)}$ by

$$T_\pi^{(\lambda)}\mathbf{q} \stackrel{\text{def}}{=\!=} (\mathbf{I}-\gamma\lambda\mathbf{P}_\pi)^{-1}[\mathbf{r}+\gamma(1-\lambda)\mathbf{P}_\pi\mathbf{q}]$$

for all $\mathbf{q}\in\mathbb{R}^{|\mathcal{S}|\cdot|\mathcal{A}|}$, where $\mathbf{I}$ is the identity matrix. Then q_π satisfies the multi-step Bellman equation $\mathbf{q}_\pi = T_\pi^{(\lambda)}\mathbf{q}_\pi$, where $\mathbf{q}_\pi$ stands for the action-value function in vector notation.

We approximate the action-value function as a linear function of some given features of state-action pairs: $q_\pi(s,a)\approx\boldsymbol{\theta}^\top\boldsymbol{\phi}(s,a)$, where $\boldsymbol{\theta}\in\mathbb{R}^n$ is the parameter vector to be estimated and $\boldsymbol{\phi}(s,a)\in\mathbb{R}^n$ is the feature vector for state s and action a. In matrix notation we write this approximation as $\mathbf{q}_\pi\approx\boldsymbol{\Phi}\boldsymbol{\theta}$, where $\boldsymbol{\Phi}\in\mathbb{R}^{|\mathcal{S}|\cdot|\mathcal{A}|\times n}$ is the feature matrix with the rows being the feature vectors for different state-action pairs: $[\boldsymbol{\Phi}]_{sa,:}=\boldsymbol{\phi}(s,a)^\top$.

The multi-step solution to the off-policy learning problem with function approximation can be found by solving the fixed point equation: $\boldsymbol{\Phi}\boldsymbol{\theta}=\boldsymbol{\Pi}_\mu T_\pi^{(\lambda)}\boldsymbol{\Phi}\boldsymbol{\theta}$ (when it has a solution), where $\boldsymbol{\Pi}_\mu \stackrel{\text{def}}{=\!=} \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\mathbf{D}_\mu\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{D}_\mu$ is the projection matrix with $\mathbf{D}_\mu\in\mathbb{R}^{|\mathcal{S}|\cdot|\mathcal{A}|\times|\mathcal{S}|\cdot|\mathcal{A}|}$ being a diagonal matrix with diagonal elements $d_\mu(s,a)$. The Mean Squared Projected Bellman Error (MSPBE) corresponding to this equation is given by:

$$J(\boldsymbol{\theta}) = \left\|\boldsymbol{\Pi}_\mu T_\pi^{(\lambda)}\boldsymbol{\Phi}\boldsymbol{\theta}-\boldsymbol{\Phi}\boldsymbol{\theta}\right\|^2_{\mathbf{D}_\mu}, \tag{8.2}$$

where $\|\mathbf{q}\|^2_{\mathbf{D}_\mu}=\mathbf{q}^\top\mathbf{D}_\mu\mathbf{q}$, for any $\mathbf{q}\in\mathbb{R}^{|\mathcal{S}|\cdot|\mathcal{A}|}$. The multi-step asymptotic TD solution associated with the fixed-point equation and the above MSPBE can be expressed as $\boldsymbol{\theta}_\infty \stackrel{\text{def}}{=\!=} \mathbf{A}^{-1}\mathbf{b}$, when $\mathbf{A}$ is an invertible matrix, and $\mathbf{A}$ and $\mathbf{b}$ are given by

$$\mathbf{A} \stackrel{\text{def}}{=\!=} \boldsymbol{\Phi}^\top\mathbf{D}_\mu\left(\mathbf{I}-\gamma\lambda\mathbf{P}_\pi\right)^{-1}\left(\mathbf{I}-\gamma\mathbf{P}_\pi\right)\boldsymbol{\Phi}, \tag{8.3}$$

$$\mathbf{b} \stackrel{\text{def}}{=\!=} \boldsymbol{\Phi}^\top\mathbf{D}_\mu\left(\mathbf{I}-\gamma\lambda\mathbf{P}_\pi\right)^{-1}\mathbf{r}. \tag{8.4}$$

8.2 The Advantage of Multi-step Learning

Under the rubric of temporal-difference learning fall a broad spectrum of methods. On one end of the spectrum, we have one-step methods that fully bootstrap using estimates of the next state and use only the immediate rewards as samples. On the other end of the spectrum, we have Monte Carlo methods that do not bootstrap and rather use all

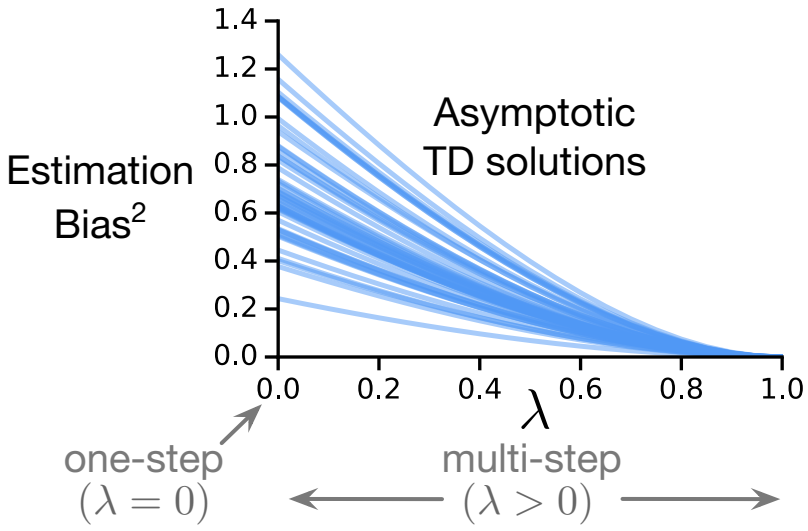

Figure 8.1: Multi-step solutions are generally superior to one-step solutions, as estimation bias typically goes monotonically to zero with increasing λ, shown here for 50 randomly constructed MDPs. In these MDPs, we used 100 states, 5 actions, and 40 features. The rewards, probabilities, and feature values (binary) were chosen uniformly randomly.

future rewards for making updates. Many multi-step learning algorithms incorporate this full spectrum and can vary smoothly between one-step and Monte Carlo updates using the bootstrapping parameter λ. Here $1-\lambda$ determines the degree to which bootstrapping is used in the algorithm. With $\lambda = 0$, these algorithms achieve one-step TD updates, whereas with $\lambda = 1$, they effectively achieve Monte Carlo updates. To contrast with one-step learning, multi-step learning is generally viewed as learning with $\lambda > 0$ in TD methods.

Multi-step learning impacts the efficiency of estimation in two ways. First, it allows more efficient estimation compared to one-step learning with a finite amount of samples. One-step learning uses the minimal amount of samples, have relatively less variance compared to Monte Carlo updates, but produces biased estimates. Typically, with a finite amount of samples, a value of λ between 0 and 1 reduces the estimation error the most (Sutton & Barto 1998).

Second, when function approximation is used and q_π does not lie in the approximation subspace, multi-step learning can produce superior asymptotic solutions compared to one-step learning. As λ increases, the multi-step Bellman operator approaches the constant operator that maps every q to q_π. This in general leads to better approximations, as suggested by the monotonically improving error bound of asymptotic solutions (Tsitsiklis & Van Roy 1997) in the on-policy case, and as we demonstrate for the off-policy case in Figure 8.1.

Although multi-step learning is desirable with function approximation, it is more dif-

ficult in the off-policy case where the detrimental effect of importance sampling is most pronounced. For this reason, off-policy learning without importance-sampling ratios is a naturally appealing and desirable solution to this problem. Prior works on off-policy learning without the ratios (e.g., Precup et al. 2000, Harutyunyan et al. 2016) are given in the lookup table case where the benefit of multi-step learning does not show up, because regardless of λ, the asymptotic solution is q_π. It is in the case of function approximation that multi-step off-policy learning without importance-sampling ratios is most needed.

8.3 Multi-step Off-policy Learning with Importance-Sampling Ratios

To setup the stage for our work, we describe in this section the canonical multi-step off-policy learning update with importance-sampling ratios, and how the ratios introduce variance in off-policy temporal-difference (TD) updates. A TD update is generally constructed based on stochastic approximation methods, where the target of the update is based on returns. Here we consider the off-line update for off-policy TD learning. Although not practical for implementation, off-line updates are useful for deriving a multi-step Bellman operator and a practically implementable algorithm. An off-line TD update for off-policy action-value estimation based on multi-step returns can be defined as:

$$\Delta\boldsymbol{\theta}_t = \alpha_t \left(G_t^\lambda - \boldsymbol{\theta}^\top \boldsymbol{\phi}_t \right) \boldsymbol{\phi}_t, \tag{8.5}$$

where $\alpha > 0$ is the step-size parameter, and $\boldsymbol{\theta}$ is a fixed weight vector. Here, G_t^λ is the multi-step target, known as λ-return, defined as the sum of TD errors weighted by powers of $\gamma\lambda$ and products of importance-sampling ratios:

$$G_t^\lambda \stackrel{\text{def}}{=\!=} \sum_{n=t}^{\infty} (\gamma\lambda)^{n-t} \rho_{t+1}^n \delta_n + \boldsymbol{\theta}^\top \boldsymbol{\phi}_t. \tag{8.6}$$

The TD error δ_t is defined as $\delta_t \stackrel{\text{def}}{=\!=} R_{t+1} + \gamma \boldsymbol{\theta}^\top \bar{\boldsymbol{\phi}}_{t+1} - \boldsymbol{\theta}^\top \boldsymbol{\phi}_t$, with $\bar{\boldsymbol{\phi}}_t \stackrel{\text{def}}{=\!=} \sum_a \pi(a|S_t)^\top \boldsymbol{\phi}(S_t, a)$. The term $\rho_t^n \stackrel{\text{def}}{=\!=} \Pi_{i=t}^n \rho_i$ is a product of importance-sampling ratios $\rho_t \stackrel{\text{def}}{=\!=} \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}$, which accounts for the discrepancy due to using the behavior policy instead of the target policy throughout the trajectory. Note that, the update defined by (8.5) is a forward-view update, that is, it uses samples that only become available in the future from the time the state of the updated estimate is visited. We call this update *the off-policy $Q(\lambda)$ update*. It can be shown that the asymptotic multi-step solution corresponding to off-policy Q(λ) is given by $\boldsymbol{\theta}_\infty = \mathbf{A}^{-1}\mathbf{b}$, (8.3), and (8.4), when $\mathbf{A}$ is invertible. All existing multi-step off-policy algorithms with importance sampling are of this form or a variant.

When $\lambda = 0$, no importance-sampling ratios are involved, and this update reduces to that of off-policy expected Sarsa (Sutton & Barto 1998, Sutton et al. 2014, van Hasselt 2011). This one-step update is also closely related to the one-step Q-learning update, where a greedy nonstationary target policy is used instead.

The importance-sampling ratios play a role when $\lambda > 0$, and their influence is greater with larger λ, including the detrimental impact on variance. The product ρ_1^n of off-policy $\mathrm{Q}(\lambda)$ in (8.6) can become as large as $\frac{1}{(\min_{s,a} \mu(a|s))^n}$. Such an exponential growth, when occurred even momentarily, can have large impact on the variance of the estimate. If the value of λ is small or very close to zero, the large variance ensuing from the product may be avoided, but it would also be devoid of much of the benefits of multi-step learning.

8.4 Avoiding Importance-Sampling Ratios

We introduce the idea of action-dependent bootstrapping and how it can be used to avoid importance-sampling ratios in off-policy estimates. For that, first we introduce an action-dependent bootstrapping parameter $\lambda(s, a) \in [0, 1]$, which is allowed to vary between different state-action pairs. A closely related idea is state-dependent bootstrapping used by Sutton and Singh (1994) and Sutton et al. (2014) for state-value estimation, and by Maei and Sutton (2010) for action-value estimation. In those works, the degree of bootstrapping was allowed to vary from one state to another by the state-dependent bootstrapping parameter $\lambda(s) \in [0, 1]$ but was not used as a device to reduce the estimation variance.

The variability of the parameter $\lambda(s, a)$ can be utilized algorithmically on a moment-by-moment basis to absorb the detrimental effect of importance sampling and in general to control the impact of importance sampling. Let us use the notational shorthand $\lambda_t \stackrel{\text{def}}{=\!=} \lambda(S_t, A_t)$, and define a new λ-return by replacing the constant λ in (8.6) with variable $\lambda(s, a)$:

$$G_t^\lambda = \sum_{n=t}^{\infty} \gamma^{n-t} \lambda_{t+1}^n \rho_{t+1}^n \delta_n + \boldsymbol{\theta}^\top \boldsymbol{\phi}_t, \tag{8.7}$$

where $\lambda_t^n \stackrel{\text{def}}{=\!=} \Pi_{i=t}^n \lambda_i$. Notice that each importance sampling ratio in (8.7) is factored with a corresponding bootstrapping parameter: $\lambda_t \rho_t$. We can mitigate an explicit presence of importance sampling ratios by setting the action-dependent bootstrapping parameter $\lambda(s, a)$ in the following way:

$$\lambda(s, a) = \nu(\psi, s, a)\mu(a|s), \qquad \nu(\psi, s, a) \stackrel{\text{def}}{=\!=} \min\left(\psi, \frac{1}{\max\left(\mu(a|s), \pi(a|s)\right)}\right) \tag{8.8}$$

where $\psi \geq 0$ is a constant. Note that $\nu(\psi, s, a)$ is upper-bounded by $\psi_{\max}$, which is defined as follows:

$$\psi_{\max} \stackrel{\text{def}}{=\!=} \frac{1}{\min_{s,a} \max\left(\mu(a|s), \pi(a|s)\right)}. \tag{8.9}$$

The product $\lambda_t \rho_t$ can then be rewritten as: $\lambda_t \rho_t = \nu(\psi, S_t, A_t)\mu_t \frac{\pi_t}{\mu_t} = \nu(\psi, S_t, A_t)\pi_t$, dispelling an explicit presence of importance sampling ratios from the update. It is easy to see that, under our proposed scheme, the effective bootstrapping parameter is upper bounded by 1: $\lambda_t \leq 1$, and at the same time all the products are also upper bounded by one: $\lambda_t^n \rho_t^n \leq 1$, largely reducing variance.

To understand how ψ influences $\lambda(s,a)$ let us use the following example, where there are only one state and three actions $\{1,2,3\}$ available. The behavior policy probabilities are $[0.2, 0.3, 0.5]$ and the target policy probabilities are $[0.2, 0.4, 0.4]$ for the three actions, respectively. Figure 8.2 shows how the action-dependent bootstrapping parameter λ for different actions change as ψ is increased from 0 to $\psi_{\max}$. Initially, the bootstrapping parameter λ increased linearly for all actions at a different rate depending on their corresponding behavior policy probabilities. The min in the factor ν comes into effect with $\psi > \psi_0 \stackrel{\text{def}}{=\!=} \frac{1}{\max_{s,a} \max(\mu(a|s), \pi(a|s))}$, and the largest λ at $\psi = \psi_0$ gets capped first. Eventually, with large enough ψ, that is, $\psi \geq \psi_{\max}$, all λs get capped.

Algorithms with constant λ are typically studied in terms of their parameters by varying λ between $[0,1]$, which would not be possible for an algorithm based on the above scheme as λ is not a constant any more. For our scheme, the constant tunable parameter is ψ, which has three pivotal values: $[0, \psi_0, \psi_{\max}]$. For parameter studies, it would be convenient if ψ is scaled to another tunable parameter between $[0,1]$. But in that case, we have to make a decision on what value ψ_0 should transform to. In the absence of a clear sense of it, a default choice would be to transform ψ_0 to 0.5. One such scaling is where we set ψ as a function of another constant $\zeta \geq 0$ in the following way:

$$\psi(\zeta) = 2\zeta\psi_0 + (2\zeta - 1)^+ \times (\psi_{\max} - 2\psi_0)\,, \tag{8.10}$$

and then vary ζ between $[0,1]$ as one would vary λ. Here $(x)^+ = \max(0, x)$. In this case, $\zeta = 0, 0.5$, and 1 correspond to $\psi(\zeta) = 0, \psi_0$, and $\psi_{\max}$, respectively. Note that ζ can be written conversely in terms of ψ as follows:

$$\zeta = (\psi - \psi_0)^+ \cdot \frac{2\psi_0 - \psi_{\max}}{2\,(\psi_{\max} - \psi_0)\,\psi_0} + \frac{\psi}{2\psi_0}. \tag{8.11}$$

The top margin of Figure 8.2 shows an alternate x-axis in terms of ζ.

To form an update using this proposed modification to λ, let us use the following notational shorthands,

$$\nu_\zeta(s,a) \stackrel{\text{def}}{=\!=} \nu(\psi(\zeta), s, a), \qquad \nu_{\zeta,t} \stackrel{\text{def}}{=\!=} \nu_\zeta(S_t, A_t), \tag{8.12}$$

$$\lambda_\zeta(s,a) \stackrel{\text{def}}{=\!=} \nu(\psi(\zeta), s, a)\mu(a|s), \qquad \lambda_{\zeta,t} \stackrel{\text{def}}{=\!=} \lambda_\zeta(S_t, A_t). \tag{8.13}$$

We use H_t^ζ to denote the λ-return defined by (8.7) with the bootstrapping parameter set according to λ_ζ:

$$H_t^\zeta = \sum_{n=t}^{\infty} \gamma^{n-t} \lambda_{\zeta,t+1}^n \rho_{t+1}^n \delta_n + \boldsymbol{\theta}^\top \boldsymbol{\phi}_t = \sum_{n=t}^{\infty} \gamma^{n-t} \nu_{\zeta,t+1}^n \pi_{t+1}^n \delta_n + \boldsymbol{\theta}^\top \boldsymbol{\phi}_t, \tag{8.14}$$

where $\lambda_{\zeta,t}^n \stackrel{\text{def}}{=\!=} \Pi_{i=t}^n \lambda_{\zeta,i}$, $\nu_{\zeta,t}^n \stackrel{\text{def}}{=\!=} \Pi_{i=t}^n \nu_{\zeta,i}$, and $\pi_t^n \stackrel{\text{def}}{=\!=} \Pi_{i=t}^n \pi_i$. The off-line forward-view update with H_t^ζ as the target can be written as:

$$\Delta\boldsymbol{\theta}_t = \alpha_t \left(H_t^\zeta - \boldsymbol{\theta}^\top \boldsymbol{\phi}_t\right) \boldsymbol{\phi}_t. \tag{8.15}$$

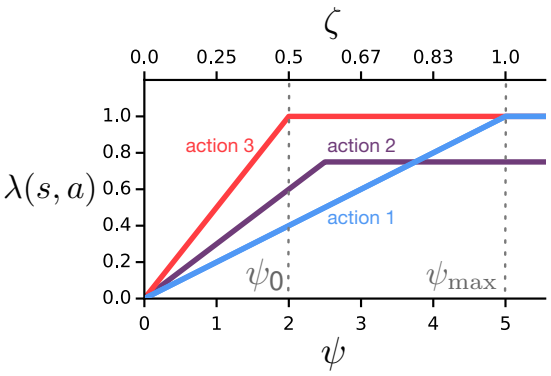

Figure 8.2: The effect of ψ and ζ on $\lambda(s, a)$ for three different actions under the action-dependent bootstrapping scheme. As ψ is increased, the parameter λ for different actions increase at a different rate and get capped for different values of ψ.

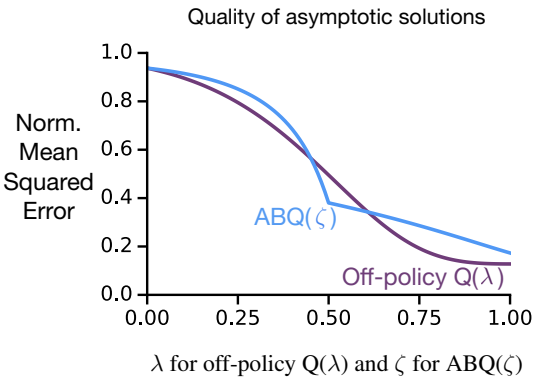

Figure 8.3: The effect of λ on off-policy Q(λ) solutions and ζ on ABQ(ζ) solutions. Multi-step ($\lambda > 0$) off-policy Q(λ) solutions are superior to the one-step ($\lambda = 0$) solution. ABQ(ζ) solutions can also achieve a similar multi-step advantage with $\zeta > 0$.

The asymptotic solution corresponding to this update, which we call the *ABQ*(ζ) solution, is $\boldsymbol{\theta}_\infty^\zeta \stackrel{\text{def}}{=} \mathbf{A}_\zeta{}^{-1}\mathbf{b}_\zeta$ with

$$\mathbf{A}_\zeta \stackrel{\text{def}}{=} \boldsymbol{\Phi}^\top \mathbf{D}_\mu \left(\mathbf{I} - \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta\right)^{-1} \left(\mathbf{I} - \gamma \mathbf{P}_\pi\right) \boldsymbol{\Phi}, \tag{8.16}$$

$$\mathbf{b}_\zeta \stackrel{\text{def}}{=} \boldsymbol{\Phi}^\top \mathbf{D}_\mu \left(\mathbf{I} - \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta\right)^{-1} \mathbf{r}, \tag{8.17}$$

assuming $\mathbf{A}_\zeta$ is invertible.

This is a multi-step solution when the bootstrapping parameter $\lambda_\zeta(s, a)$ does not uniformly reduce to zero. The drawback of this scheme is that we cannot achieve $\lambda_\zeta(s, a) = 1$ for all state-action pairs, which would produce the off-policy Monte Carlo solution. It is the cost of avoiding importance-sampling ratios together with its large variance issue.

To illustrate that ABQ(ζ) can achieve multi-step learning, we used a two-state off-policy task similar to the off-policy task by Sutton et al. (2016). In this task, there were two states each with two actions, *left* and *right*, leading to one of the two states deterministically. More specifically, $p(1|1, \mathit{left}) = p(2|1, \mathit{right}) = p(1|2, \mathit{left}) = p(2|2, \mathit{right}) = 1$. There is a deterministic nonzero reward $r(2, \mathit{right}) = +1$; all other transitions have reward zero. The discount factor γ was 0.9. The feature vectors were set as $\boldsymbol{\phi}(1, \mathit{left}) = \boldsymbol{\phi}(1, \mathit{right}) = 1$ and $\boldsymbol{\phi}(2, \mathit{left}) = \boldsymbol{\phi}(2, \mathit{right}) = 2$. The behavior policy was chosen as $\mu(\mathit{right}|1) = \mu(\mathit{left}|2) = 0.9$ to break away from the uniform distribution of the original problem. The target policy was chosen as $\pi(\mathit{right}|\cdot) = 0.9$.

We produced different asymptotic solutions defined by (8.16) and (8.17), choosing different constant ζ between 0 and $\zeta_{\max}$. We compared ABQ(ζ) solutions with off-policy Q(λ) solutions in terms of the Mean Squared Error (MSE) $\|\boldsymbol{\Phi}\boldsymbol{\theta} - \mathbf{q}_\pi\|^2_{\mathbf{D}_\mu}$ normalized by $\|\mathbf{q}_\pi\|^2_{\mathbf{D}_\mu}$.

The results are given in 8.3. Off-policy Q(λ) solutions with $\lambda > 0$ in this task are substantially better than the one-step solution produced with $\lambda = 0$. The ABQ(ζ) solutions cannot be as good as the off-policy Q(1) solution, as we already anticipated, but much of the benefits of multi-step off-policy solutions can be attained by choosing a large value of ζ. Although the tunable parameter ζ of ABQ was set to be a constant, the effective bootstrapping parameter $\lambda_\zeta(s, a)$ was different for different state-action pairs. Therefore, ABQ solutions cannot be obtained simply by rescaling the constant λ of off-policy Q(λ).

8.5 The ABQ(ζ) Algorithm with Gradient Correction and Scalable Updates

In this section, we develop a computationally scalable and stable algorithm corresponding to the ABQ(ζ) solution. The update (8.15) given earlier cannot be computed in a scalable manner, because forward-view updates require increasing amount of computation as time progresses. Moroever, off-policy algorithms with bootstrapping and functions approximation may be unstable (Sutton & Barto 1998), unless machinery for ensuring stability is included. Our goal is to develop a stable update corresponding to ABQ(ζ) solutions while keeping it free from importance-sampling ratios.

First, we produce the equivalent backward view of (8.15) so that the updates can be implemented without forming explicitly the λ-return.

Using the forward-view update given by (8.15), the total update can be given by:

$$\sum_{t=0}^{\infty} \Delta\boldsymbol{\theta}_t = \sum_{t=0}^{\infty} \alpha_t \left(H_t^\zeta - \boldsymbol{\theta}^\top \boldsymbol{\phi}_t \right) \boldsymbol{\phi}_t \tag{8.18}$$

$$= \sum_{t=0}^{\infty} \sum_{n=t}^{\infty} \gamma^{n-t} \nu_{\zeta,t+1}^{n} \pi_{t+1}^{n} \delta_n \boldsymbol{\phi}_t \tag{8.19}$$

$$= \sum_{t=0}^{\infty} \alpha_t \sum_{n=0}^{t} \gamma^{t-n} \nu_{\zeta,n+1}^{t} \pi_{n+1}^{t} \delta_t \boldsymbol{\phi}_n \tag{8.20}$$

$$= \sum_{t=0}^{\infty} \alpha_t \delta_t \underbrace{\sum_{n=0}^{t} \gamma^{t-n} \nu_{\zeta,n+1}^{t} \pi_{n+1}^{t} \boldsymbol{\phi}_n}_{\mathbf{e}_t} \tag{8.21}$$

$$= \sum_{t=0}^{\infty} \alpha_t \delta_t \mathbf{e}_t. \tag{8.22}$$

Therefore, the backward-view update can be written as:

$$\Delta\boldsymbol{\theta}_t^B = \alpha_t \delta_t \mathbf{e}_t. \tag{8.23}$$

The eligibility trace vector $\mathbf{e}_t \in \mathbb{R}^n$ can be written recursively as:

$$\mathbf{e}_t = \sum_{n=0}^{t} \gamma^{t-n} \nu_{\zeta,n+1}^{t} \pi_{n+1}^{t} \boldsymbol{\phi}_n \tag{8.24}$$

$$= \gamma \nu_{\zeta,t} \pi_t \sum_{n=0}^{t-1} \gamma^{t-n-1} \nu_{\zeta,n+1}^{t-1} \pi_{n+1}^{t-1} \boldsymbol{\phi}_n + \boldsymbol{\phi}_t \tag{8.25}$$

$$= \gamma \nu_{\zeta,t} \pi_t \mathbf{e}_{t-1} + \boldsymbol{\phi}_t. \tag{8.26}$$

Therefore, the backward view updates are given by

$$\Delta \boldsymbol{\theta}_t = \alpha_t \delta_t \mathbf{e}_t, \qquad \mathbf{e}_t = \gamma \nu_{\zeta,t} \pi_t \mathbf{e}_{t-1} + \boldsymbol{\phi}_t. \tag{8.27}$$

Here, $\mathbf{e}_t \in \mathbb{R}^n$ is an accumulating trace vector. The above backward-view update achieves equivalence with the forward-view of (8.15) only for off-line updating, which is typical for all algorithms with accumulating traces. Equivalence for online updating could also be achieved by following the approach taken by van Seijen et al. (2016), but we leave that out here for simplicity.

We take the approach proposed by Maei (2011) to develop a stable algorithm. In the following, we derive the resulting gradient corrected algorithm, which we call *the $ABQ(\zeta)$ algorithm*.

The key step in deriving a gradient-based TD algorithm, as proposed by Maei (2011), is to formulate an associated Mean Squared Projected Bellman Error (MSPBE) and produce its gradient, which can then be sampled to produce stochastic updates.

The MSPBE for the update (8.15) is given by:

$$J(\boldsymbol{\theta}) = \left\| \boldsymbol{\Pi}_\mu T_\pi^{(\boldsymbol{\Lambda}_\zeta)} \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\Phi}\boldsymbol{\theta} \right\|_{\mathbf{D}_\mu}^2 \tag{8.28}$$

$$= \left\| \boldsymbol{\Pi}_\mu \left(T_\pi^{(\boldsymbol{\Lambda}_\zeta)} \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\Phi}\boldsymbol{\theta} \right) \right\|_{\mathbf{D}_\mu}^2 \tag{8.29}$$

$$= \left(T_\pi^{(\boldsymbol{\Lambda}_\zeta)} \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\Phi}\boldsymbol{\theta} \right)^\top \boldsymbol{\Pi}_\mu^\top \mathbf{D}_\mu \boldsymbol{\Pi}_\mu \left(T_\pi^{(\boldsymbol{\Lambda}_\zeta)} \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\Phi}\boldsymbol{\theta} \right) \tag{8.30}$$

$$= \left(\boldsymbol{\Phi}^\top \mathbf{D}_\mu \left(T_\pi^{(\boldsymbol{\Lambda}_\zeta)} \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\Phi}\boldsymbol{\theta} \right) \right)^\top (\boldsymbol{\Phi}^\top \mathbf{D}_\mu \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top \mathbf{D}_\mu \left(T_\pi^{(\boldsymbol{\Lambda}_\zeta)} \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\Phi}\boldsymbol{\theta} \right). \tag{8.31}$$

Here, the Bellman operator corresponding to the bootstrapping matrix $\boldsymbol{\Lambda}_\zeta$ is defined for all $\mathbf{q} \in \mathbb{R}^{|\mathcal{S}| \cdot |\mathcal{A}|}$ as

$$T_\pi^{(\boldsymbol{\Lambda}_\zeta)} \mathbf{q} \stackrel{\text{def}}{=\!=} (\mathbf{I} - \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta)^{-1} \left[\mathbf{r} + \gamma \mathbf{P}_\pi (\mathbf{I} - \boldsymbol{\Lambda}_\zeta) \mathbf{q} \right].$$

Let $\mathbf{g} \stackrel{\text{def}}{=\!=} \boldsymbol{\Phi}^\top \mathbf{D}_\mu \left(T_\pi^{(\boldsymbol{\Lambda}_\zeta)} \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\Phi}\boldsymbol{\theta} \right) = \boldsymbol{\Phi}^\top \mathbf{D}_\mu \left((\mathbf{I} - \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta)^{-1} \left[\mathbf{r} + \gamma \mathbf{P}_\pi (\mathbf{I} - \boldsymbol{\Lambda}_\zeta) \boldsymbol{\Phi}\boldsymbol{\theta} \right] - \boldsymbol{\Phi}\boldsymbol{\theta} \right)$ (8.32)

$$= \boldsymbol{\Phi}^\top \mathbf{D}_\mu \left((\mathbf{I} - \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta)^{-1} \left[\mathbf{r} + \left((\mathbf{I} - \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta) - (\mathbf{I} - \gamma \mathbf{P}_\pi) \right) \boldsymbol{\Phi}\boldsymbol{\theta} \right] - \boldsymbol{\Phi}\boldsymbol{\theta} \right) \tag{8.33}$$

$$= \boldsymbol{\Phi}^\top \mathbf{D}_\mu \left((\mathbf{I} - \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta)^{-1} \mathbf{r} + \boldsymbol{\Phi}\boldsymbol{\theta} - (\mathbf{I} - \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta)^{-1} (\mathbf{I} - \gamma \mathbf{P}_\pi) \boldsymbol{\Phi}\boldsymbol{\theta} - \boldsymbol{\Phi}\boldsymbol{\theta} \right) \tag{8.34}$$

$$= \boldsymbol{\Phi}^\top \mathbf{D}_\mu (\mathbf{I} - \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta)^{-1} \left(\mathbf{r} - (\mathbf{I} - \gamma \mathbf{P}_\pi) \boldsymbol{\Phi}\boldsymbol{\theta} \right). \tag{8.35}$$

Also let $\mathbf{C} = (\boldsymbol{\Phi}^\top \mathbf{D}_\mu \boldsymbol{\Phi})$. Then the gradient can be written as:

$$\nabla J(\boldsymbol{\theta}) = -\left(X^\top \mathbf{D}_\mu (\mathbf{I} - \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta)^{-1} (\mathbf{I} - \gamma \mathbf{P}_\pi) \boldsymbol{\Phi} \right)^\top \mathbf{C}^{-1} \mathbf{g} \tag{8.36}$$

$$= -\left(\mathbf{\Phi}^\top \mathbf{D}_\mu (\mathbf{I} - \gamma \mathbf{P}_\pi \mathbf{\Lambda}_\zeta)^{-1} \left((\mathbf{I} - \gamma \mathbf{P}_\pi \mathbf{\Lambda}_\zeta)\, \mathbf{\Phi} - \gamma \mathbf{P}_\pi (\mathbf{I} - \mathbf{\Lambda}) \mathbf{\Phi}\right)\right)^\top \mathbf{C}^{-1} \mathbf{g} \tag{8.37}$$

$$= -\left(\mathbf{\Phi}^\top \mathbf{D}_\mu \mathbf{\Phi} - \gamma \mathbf{\Phi}^\top \mathbf{D}_\mu \left(\mathbf{I} - \gamma \mathbf{P}_\pi \mathbf{\Lambda}_\zeta\right)^{-1} \mathbf{P}_\pi (\mathbf{I} - \mathbf{\Lambda}) \mathbf{\Phi}\right)^\top \mathbf{C}^{-1} \mathbf{g} \tag{8.38}$$

$$= -\left(\mathbf{g} - \gamma \left(\underbrace{\mathbf{\Phi}^\top \mathbf{D}_\mu \left(\mathbf{I} - \gamma \mathbf{P}_\pi \mathbf{\Lambda}_\zeta\right)^{-1} \mathbf{P}_\pi (\mathbf{I} - \mathbf{\Lambda}) \mathbf{\Phi}}_{\mathbf{H}}\right)^\top \mathbf{C}^{-1} \mathbf{g}\right) \tag{8.39}$$

$$= -\left(\mathbf{g} - \gamma \mathbf{H}^\top \mathbf{C}^{-1} \mathbf{g}\right). \tag{8.40}$$

So if we know the gradient, the gradient-descent step would be to add to the parameter vector

$$-\alpha_t \nabla J(\boldsymbol{\theta}) = \alpha_t \left(\mathbf{g} - \gamma \mathbf{H}^\top \mathbf{C}^{-1} \mathbf{g}\right). \tag{8.41}$$

Now to derive the stochastic updates for ABQ(ζ), let us consider a double-ended *stationary* Markov chain induced by μ, $\{\ldots, (S_{-2}, A_{-2}, R_{-1}), (S_{-1}, A_{-1}, R_0), (S_0, A_0, R_1), (S_1, A_1, R_2), \ldots\}$. Let E_0 denote expectation with respect to the probability distribution of this stationary Markov chain. Fix t to be any integer. For the first term $\mathbf{g}$ in $\nabla J(\boldsymbol{\theta})$, we can write:

$$\mathbf{g} = \mathbf{\Phi}^\top \mathbf{D}_\mu (\mathbf{I} - \gamma \mathbf{P}_\pi \mathbf{\Lambda}_\zeta)^{-1} \left(\mathbf{r} - (\mathbf{I} - \gamma \mathbf{P}_\pi) \mathbf{\Phi} \boldsymbol{\theta}\right) \tag{8.42}$$

$$= \mathbf{\Phi}^\top \mathbf{D}_\mu \left(\mathbf{I} - \gamma \mathbf{P}_\pi \mathbf{\Lambda}_\zeta\right)^{-1} \mathbf{r} - \mathbf{\Phi}^\top \mathbf{D}_\mu \left(\mathbf{I} - \gamma \mathbf{P}_\pi \mathbf{\Lambda}_\zeta\right)^{-1} (\mathbf{I} - \gamma \mathbf{P}_\pi) \mathbf{\Phi} \boldsymbol{\theta} \tag{8.43}$$

$$= \mathbf{b}_\zeta - \mathbf{A}_\zeta \boldsymbol{\theta}; \qquad (\mathbf{A}_\zeta \text{ and } \mathbf{b}_\zeta \text{ as defined by (8.16) and (8.17) }) \tag{8.44}$$

$$= \mathrm{E}_0 \left[\left(H_t^\zeta - \boldsymbol{\theta}^\top \boldsymbol{\phi}_t\right) \boldsymbol{\phi}_t\right] \tag{8.45}$$

$$= \mathrm{E}_0 \left[\sum_{n=t}^{\infty} \gamma^{n-t} \nu_{\zeta,t+1}^n \pi_{t+1}^n \delta_n \boldsymbol{\phi}_t\right] \tag{8.46}$$

$$= \mathrm{E}_0 \left[\delta_t \boldsymbol{\phi}_t + \sum_{n=t+1}^{\infty} \gamma^{n-t} \nu_{\zeta,t+1}^n \pi_{t+1}^n \delta_n \boldsymbol{\phi}_t\right] \tag{8.47}$$

$$= \mathrm{E}_0 \left[\delta_t \boldsymbol{\phi}_t + \sum_{n=t}^{\infty} \gamma^{n-(t-1)} \nu_{\zeta,t}^n \pi_t^n \delta_n \boldsymbol{\phi}_{t-1}\right]; \quad \text{shifting indices and using stationarity} \tag{8.48}$$

$$= \mathrm{E}_0 \left[\delta_t \boldsymbol{\phi}_t + \gamma \nu_{\zeta,t} \pi_t \sum_{n=t}^{\infty} \gamma^{n-t} \nu_{\zeta,t+1}^n \pi_{t+1}^n \delta_n \boldsymbol{\phi}_{t-1}\right] \tag{8.49}$$

$$= \mathrm{E}_0 \left[\delta_t \boldsymbol{\phi}_t + \gamma \nu_{\zeta,t} \pi_t \left(\delta_t \boldsymbol{\phi}_{t-1} + \sum_{n=t+1}^{\infty} \gamma^{n-t} \nu_{\zeta,t+1}^n \pi_{t+1}^n \delta_n \boldsymbol{\phi}_{t-1}\right)\right] \tag{8.50}$$

$$= \mathrm{E}_0 \left[\delta_t \left(\boldsymbol{\phi}_t + \gamma \nu_{\zeta,t} \pi_t \boldsymbol{\phi}_{t-1} + \cdots\right)\right]; \qquad \text{shifting indices and using stationarity} \tag{8.51}$$

$$= \mathrm{E}_0 \left[\delta_t \mathbf{e}_t\right], \tag{8.52}$$

where $\mathbf{e}_t$ is a well-defined random variable and can be written recursively as:

$$\mathbf{e}_t = \boldsymbol{\phi}_t + \gamma \nu_{\zeta,t} \pi_t \boldsymbol{\phi}_{t-1} + \cdots = \boldsymbol{\phi}_t + \gamma \nu_{\zeta,t} \pi_t \mathbf{e}_{t-1}. \tag{8.53}$$

Similarly, we can also express the term $\mathbf{H}$ in $\nabla J(\boldsymbol{\theta})$ in expectation form. Let us define

$$\tilde{\boldsymbol{\phi}}_t = \sum_a \lambda_\zeta(S_t, a)\pi(a|S_t)\boldsymbol{\phi}(S_t, a). \tag{8.54}$$

Then we can write:

$$\begin{aligned}
\mathbf{H} &= X^\top \mathbf{D}_\mu \left(\mathbf{I} - \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta\right)^{-1} \mathbf{P}_\pi (\mathbf{I} - \boldsymbol{\Lambda}_\zeta) X && (8.55)\\
&= X^\top \mathbf{D}_\mu \mathbf{P}_\pi (\mathbf{I} - \boldsymbol{\Lambda}_\zeta) X + X^\top \mathbf{D}_\mu \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta \mathbf{P}_\pi (\mathbf{I} - \boldsymbol{\Lambda}_\zeta) X + \cdots && (8.56)\\
&= \sum_{s,a} d_\mu(s,a)\boldsymbol{\phi}(s,a) \sum_{s'a'} p(s'|s,a)\pi(a'|s')(1 - \lambda_\zeta(s',a'))\boldsymbol{\phi}(s',a')^\top && (8.57)\\
&\quad + X^\top \mathbf{D}_\mu \gamma \mathbf{P}_\pi \boldsymbol{\Lambda}_\zeta \mathbf{P}_\pi (\mathbf{I} - \boldsymbol{\Lambda}_\zeta) X + \cdots && (8.58)\\
&= \mathrm{E}_0 \left[\boldsymbol{\phi}_t \left(\bar{\boldsymbol{\phi}}_{t+1} - \tilde{\boldsymbol{\phi}}_{t+1} \right)^\top \right] && (8.59)\\
&\quad + \gamma \sum_{s,a} d_\mu(s,a)\boldsymbol{\phi}(s,a) \sum_{s',a'} p(s'|s,a)\pi(a'|s')\zeta(a'|s')\mu(a'|s') && (8.60)\\
&\quad \times \sum_{s''a''} p(s''|s',a')\pi(a''|s'')(1 - \lambda_\zeta(s'',a''))\boldsymbol{\phi}(s'',a'')^\top + \cdots && (8.61)\\
&= \mathrm{E}_0 \left[\boldsymbol{\phi}_t \left(\bar{\boldsymbol{\phi}}_{t+1} - \tilde{\boldsymbol{\phi}}_{t+1} \right)^\top \right] && (8.62)\\
&\quad + \mathrm{E}_0 \left[\gamma \nu_{\zeta,t+1} \pi_{t+1} \boldsymbol{\phi}_t \left(\bar{\boldsymbol{\phi}}_{t+2} - \tilde{\boldsymbol{\phi}}_{t+2} \right)^\top \right] + \cdots; \quad \text{shifting indices and using stationarity} && (8.63)\\
&= \mathrm{E}_0 \left[\boldsymbol{\phi}_t \left(\bar{\boldsymbol{\phi}}_{t+1} - \tilde{\boldsymbol{\phi}}_{t+1} \right)^\top \right] && (8.64)\\
&\quad + \mathrm{E}_0 \left[\gamma \nu_{\zeta,t} \pi_t \boldsymbol{\phi}_{t-1} \left(\bar{\boldsymbol{\phi}}_{t+1} - \tilde{\boldsymbol{\phi}}_{t+1} \right)^\top \right] + \cdots; \quad \text{shifting indices and using stationarity} && (8.65)\\
&= \mathrm{E}_0 \left[(\boldsymbol{\phi}_t + \gamma \nu_{\zeta,t} \pi_t \boldsymbol{\phi}_{t-1} + \cdots) \left(\bar{\boldsymbol{\phi}}_{t+1} - \tilde{\boldsymbol{\phi}}_{t+1} \right)^\top) \right] && (8.66)\\
&= \mathrm{E}_0 \left[\mathbf{e}_t \left(\bar{\boldsymbol{\phi}}_{t+1} - \tilde{\boldsymbol{\phi}}_{t+1} \right)^\top \right]. && (8.67)
\end{aligned}$$

Therefore, a stochastic update corresponding to the expected gradient-descent update can be written as:

$$\Delta\boldsymbol{\theta} = \alpha_t \left(\delta_t \mathbf{e}_t - \gamma \left(\bar{\boldsymbol{\phi}}_{t+1} - \tilde{\boldsymbol{\phi}}_{t+1} \right) \mathbf{e}_t^\top \mathbf{C}^{-1} \mathbf{g} \right). \tag{8.68}$$

The vector $\mathbf{C}^{-1}\mathbf{g}$ can be estimated from samples by LMS but at a faster time scale and with a larger step-size parameter β:

$$\Delta\mathbf{h} = \beta_t \left(\delta_t \mathbf{e}_t - \mathbf{h}^\top \boldsymbol{\phi}_t \boldsymbol{\phi}_t \right). \tag{8.69}$$

It can be shown that with $\boldsymbol{\theta}$ held fixed, under standard diminishing step-size rules for β_t, the $\{\mathbf{h}_t\}$ produced by the above updates converges to

$$\mathbf{h}_\infty = \left(\mathrm{E}_0 \left[\boldsymbol{\phi}_t \boldsymbol{\phi}_t^\top \right] \right)^{-1} \mathrm{E}_0 \left[\delta_t \mathbf{e}_t \right] \tag{8.70}$$

$$= \left(\boldsymbol{\Phi}^\top \mathbf{D}_\mu \boldsymbol{\Phi}\right)^{-1} \mathrm{E}_0\left[\delta_t \mathbf{e}_t\right] \tag{8.71}$$

$$= \mathbf{C}^{-1}\mathbf{g}. \tag{8.72}$$

Putting these pieces together, and also allowing $\boldsymbol{\theta}$ and $\mathbf{h}$ to vary over time, we obtain the updates of the on-line gradient-based TD algorithm, which we call ABQ(ζ):

$$\boldsymbol{\theta}_{t+1} \stackrel{\text{def}}{=} \boldsymbol{\theta}_t + \alpha_t \left(\delta_t \mathbf{e}_t - \gamma \mathbf{e}_t^\top \mathbf{h}_t \left(\bar{\boldsymbol{\phi}}_{t+1} - \tilde{\boldsymbol{\phi}}_{t+1}\right)\right), \tag{8.73}$$

$$\tilde{\boldsymbol{\phi}}_t \stackrel{\text{def}}{=} \sum_a \lambda_\zeta(S_t, a)\pi(a|S_t)\boldsymbol{\phi}(S_t, a), \tag{8.74}$$

$$\delta_t \stackrel{\text{def}}{=} R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \bar{\boldsymbol{\phi}}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t, \tag{8.75}$$

$$\mathbf{e}_t \stackrel{\text{def}}{=} \gamma \nu_{\zeta,t} \pi_t \mathbf{e}_{t-1} + \boldsymbol{\phi}_t, \tag{8.76}$$

$$\mathbf{h}_{t+1} \stackrel{\text{def}}{=} \mathbf{h}_t + \beta_t \left(\delta_t \mathbf{e}_t - \mathbf{h}_t^\top \boldsymbol{\phi}_t \boldsymbol{\phi}_t\right). \tag{8.77}$$

The iteration (8.73) carries out stochastic gradient-descent steps to minimize the MSPBE regarding the Bellman operator $T_\pi^{(\boldsymbol{\Lambda}_\zeta)}$, and it differs from (8.27) in the gradient-correction term $-\gamma \mathbf{e}_t^\top \mathbf{h}_t \left(\bar{\boldsymbol{\phi}}_{t+1} - \tilde{\boldsymbol{\phi}}_{t+1}\right)$. This correction term involves the extra vector parameter $\mathbf{h}_t$. Note that no importance-sampling ratios are needed in the update of $\mathbf{h}_t$ or in the gradient-correction term. The vector $\mathbf{h}_t$ is updated according to (8.77) at a faster timescale than $\boldsymbol{\theta}_t$ by using a second step-size parameter $\beta_t \gg \alpha_t$. Both constant and diminishing step sizes can be used. For diminishing step sizes, one can let $\alpha_t = O(1/t), \beta_t = O(1/t^c), c \in (1/2, 1)$, for instance. In practice, when stability is not a problem, smaller values of β often lead to better performance (White & White 2016).

8.6 Experimental Results

We empirically evaluate ABQ(ζ) on three policy evaluation tasks: the two-state off-policy task from Section 5, an off-policy policy evaluation adaptation of the Mountain Car domain (Sutton & Barto 1998), and the 7-star Baird's counterexample (Baird 1995, White 2015). In the first two tasks we investigated whether ABQ(ζ) can produce correct estimates with less variance compared to GQ(λ) (Maei 2011), the state-of-the-art importance sampling based algorithm for action-value estimation with function approximation. We validate the stability of ABQ(ζ) in the final task, where off-policy algorithms without a stability guarantee (e.g., off-policy Q(λ)) tend to diverge.

Although the MDP involved in the two-state task is small, off-policy algorithms may suffer severely in this task as the importance sampling ratio, once in a while, can be as large as 9. We simulated both GQ(λ) and ABQ(λ) on this task for 10000 time steps, starting with $\boldsymbol{\theta}_0 = \mathbf{0}$. We averaged the MSE $\|X\boldsymbol{\theta}_t - \mathbf{q}_\pi\|^2_{\mathbf{D}_\mu}$ for the last 5000 time steps. We further averaged this quantity over 100 independent runs. Finally, we divided this error by $\|\mathbf{q}_\pi\|^2_{\mathbf{D}_\mu}$ to obtain the normalized MSE (NMSE).

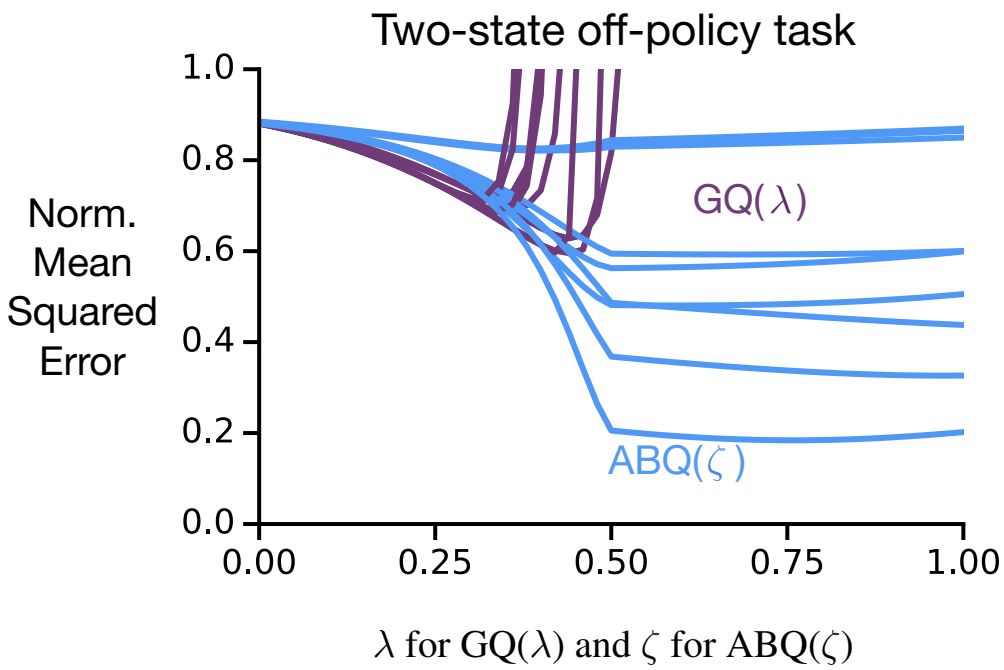

Figure 8.4: Comparison of empirical performance of GQ(λ) and ABQ(ζ) on a two-state off-policy policy evaluation task. Performance is shown in normalized mean squared error with respect to different values of λ for GQ(λ) and ζ for ABQ(ζ). Different curves are for different combinations of step-size values. GQ(λ) produces large MSE when large λ is used. ABQ(ζ) tolerates larger values of ζ and thus can better retain the benefits of multi-step learning compared to GQ(λ).

Figure 8.4 shows curves for different combinations of the step-size parameters: $\alpha \in [0.001, 0.005, 0.01]$ and $\beta \in [0.001, 0.005, 0.01]$. Performance is shown in the estimated NMSE, with the corresponding standard error for different values of λ and ζ. With $\lambda > 0.6$, the error of GQ(λ) increased sharply due to increased influence of importance sampling ratios. It clearly depicts the failure to perform effective multi-step learning by an importance sampling based algorithm when λ is large. The error of ABQ(ζ) decreased substantially for most step-size combinations as ζ is increased from 0 to 0.5, and it decreased slightly for some step-size combinations as ζ was increased up to 1. This example clearly shows that ABQ(ζ) can perform effective multi-step learning while avoiding importance sampling ratios. On the other hand, the best performance of GQ(λ) was better than ABQ(ζ) for the smallest value of the first step size (i.e., $\alpha = 0.001$). When the step size was too small, GQ(λ) in fact benefited from the occasional large scaling from importance sampling, whereas ABQ(ζ) remained conservative in its updates to be safe.

The Mountain Car domain is typically used for policy improvement tasks, but here we use it for off-policy policy evaluation. We constructed the task in such a way that the importance sampling ratios for GQ can be as large as 30, emphasizing the variance issue regarding ratios. In this task, the car starts in the vicinity of the bottom of the valley with

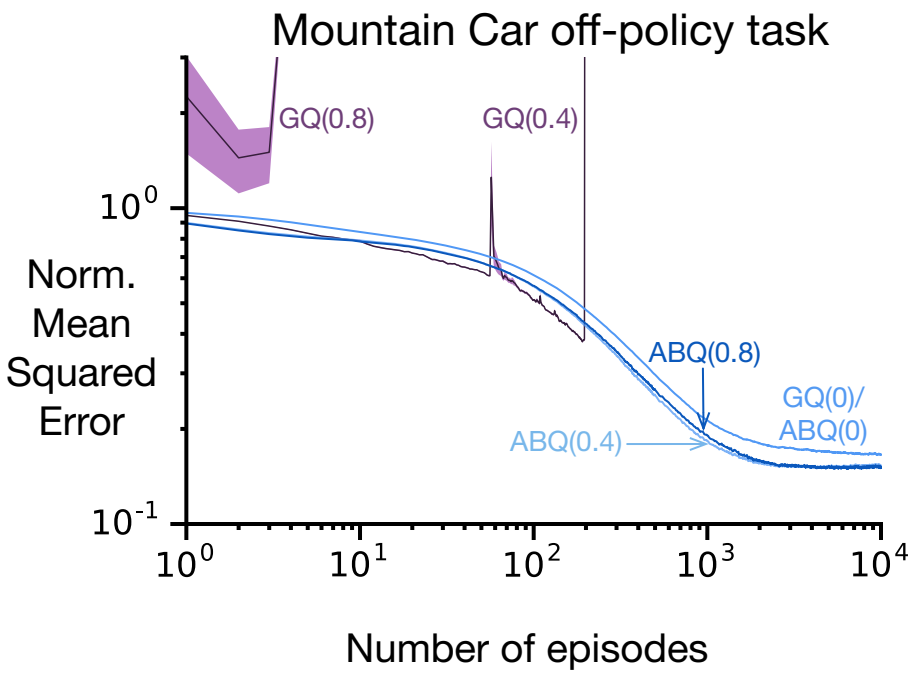

Figure 8.5: Comparison of empirical performance of GQ(λ) and ABQ(ζ) on an off-policy policy evaluation task based on the Mountain Car domain. Each curve shows how learning progresses in terms of estimated normalized mean squared error as more episodes are observed. Different learning curves are for different values of λ for GQ(λ) and ζ for ABQ(ζ). All of them are shown for a particular combination of step-size values. The spikes in GQ(λ)'s learning curves are the result of the occasional large values of importance sampling ratios, increasing the variance of the estimate for GQ(λ) as large values of λ are chosen. ABQ(ζ), on the other hand, can achieve lower mean squared error with larger values of ζ by reducing the estimation variance.

a small nonzero speed. The three actions are: *reverse throttle*, *no throttle*, and *forward throttle*. Rewards are -1 at every time step, and state transitions are deterministic. The discount factor γ is 0.999. The policies used in the experiments were based on a simple handcrafted policy, which chooses to move forward with full throttle if the velocity of the car was nonzero and toward the goal, and chooses to move away from the goal with full throttle otherwise.

Both the target policy and the behavior policy are based on this policy but choose to randomly explore the other actions differently. More specifically, when the velocity was nonzero and toward the goal, the behavior policy probabilities for the actions *reverse throttle*, *no throttle*, and *forward throttle* are $\left[\frac{1}{300}, \frac{1}{300}, \frac{298}{300}\right]$, and the target policy probabilities are $[0.1, 0.1, 0.8]$, respectively. In the other case, the behavior and the target policy probabilities are $\left[\frac{298}{300}, \frac{1}{300}, \frac{1}{300}\right]$ and $[0.8, 0.1, 0.1]$, respectively. We set the policies this way so that episodes complete under both policies in a reasonable number of time steps, while the importance sampling ratio may occasionally be as large as $0.1 \times 300 = 30$.

The feature vector for each state-action pair contained 32 features for each action,

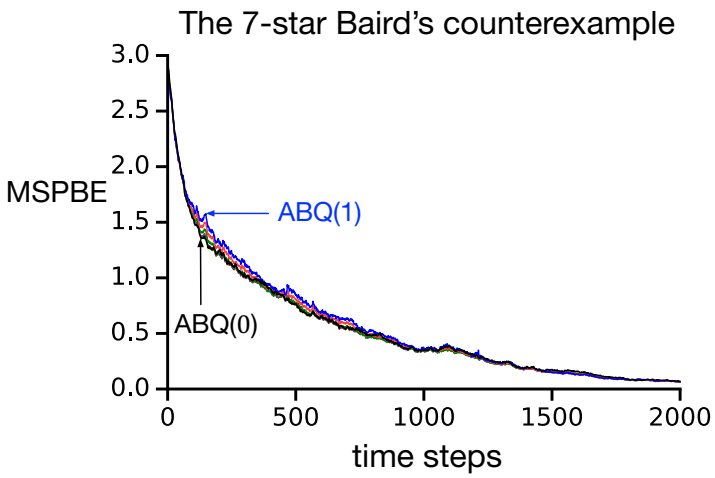

Figure 8.6: ABQ(ζ) is stable on Baird's counterexample for different values of ζ.

where only the features corresponding to the given action were nonzero, produced using tile coding with ten 4×4 tilings. Each algorithm ran on the same 100 independent sequences of samples, each consisting 10,000 episodes. The performance was measured in terms of the Mean-Squared-Error (MSE) with respect to the estimated value $\hat{\mathbf{q}} \in \mathbb{R}^{30}$ of 30 chosen state-action pairs. These pairs were chosen by running the agent under the behavior policy for 1 million time steps, restarting episodes each time termination occurs, and choosing 30 pairs uniformly randomly from the last half million time steps. The ground truth values for these 30 pairs $\hat{\mathbf{q}}$ were estimated by following the target policy 100 times from those pairs and forming the average. The mean-squared error from $\hat{\mathbf{q}}$ was normalized by $\|\hat{\mathbf{q}}\|_2^2$.

We use the mountain car off-policy task to illustrate through learning curves how λ and ζ affect GQ(λ) and ABQ(ζ), respectively. Figure 8.5 shows the learning curves of ABQ(ζ) and GQ(λ) with respect to mean squared errors for three difference values of λ and ζ: 0, 0.4, and 0.8, and a particular step-size combination: $\alpha = 0.1/($ # of active features) and $\beta = 0.0$. These learning curves are averages over 100 independent runs. The standard errors of ABQ's estimated MSE here are smaller than the width of the curves shown. ABQ achieved a significantly lower MSE with $\zeta > 0$ than with $\zeta = 0$. On the other hand, GQ performed unreliably with larger λ. With $\lambda = 0.4$, the learning curves are highly varying from each other due to occasionally having the largest ratio value 30, affecting the update at different time steps. Some learning curves even became unstable, causing the MSE to move away further and further with time, and affecting the average MSE, which explains the spikes. When $\lambda = 0.8$ was chosen, all learning curves became unstable in few steps.

In the 7-star Baird's counterexample, adopted from White (2015), step sizes were set as $\alpha = 0.05$ and $\beta = 0.1$, and the bootstrapping parameter ζ was chosen evenly between 0 and 1. The Mean Squared Projected Bellman Error (MSPBE) was estimated by averaging

over 50 runs. As shown in Figure 8.6, ABQ(ζ) performed stably with all values of ζ used. This validates empirically the gradient correction in ABQ(ζ).

8.7 Action-dependent Bootstrapping as a Framework for Off-policy Algorithms

ABQ(ζ) is a result of this new idea of varying the bootstrapping parameter in an action-dependent manner so that an explicit presence of importance sampling ratios are mitigated from the update. However, it is not the only action-dependent bootstrapping scheme one can devise. It is possible to bound the product $\lambda_t^n \rho_t^n$ by using other action-dependent bootstrapping schemes. Different schemes not only allow us to derive new algorithms, they may also be used to understand some existing algorithms better.

An algorithm closely related to ABQ is Tree Backup by Precup et al. (2000). It can be produced as a special case of ABQ(ζ), if we remove gradient correction, consider the feature vectors always to be the standard basis, and ν_ζ to be always set to a constant, instead of setting it in an action-dependent manner. In the on-policy case, the Tree Backup algorithm fails to achieve the TD(1) solution, whereas ABQ achieves it evidently with $\zeta \geq 1$. Our work is not a trivial generalization of this prior work. The Tree Backup algorithm was developed using a different intuition based on backup diagrams and was introduced only for the lookup table case. Not only does ABQ(ζ) extend the Tree Backup algorithm, but the idea of action-dependent bootstrapping also played a crucial role in deriving the ABQ(ζ) algorithm with gradient correction in a principled way.

Another related algorithm is Retrace by Munos et al. (2016), where the variance issue is approached by truncating the importance-sampling ratios. Retrace is a tabular off-policy algorithm that approaches the variance issue by truncating the importance sampling ratio. We show that such truncations can be understood as varying the action-dependent bootstrapping parameter in a particular manner, first, by constructing a forward-view update with a different action-dependent bootstrapping scheme than ABQ's, second, by deriving the asymptotic solution corresponding to that forward-view update, and third, by showing that the equivalent backward-view update is the same as the Retrace algorithm. For generality, we take these steps in the linear function approximation case and incorporate gradient corrections for stability. The resulting algorithm, we call *AB-Trace*(ζ) is a stable generalization of Retrace.

We construct a new forward-view update similar to (8.15) with $\lambda_\zeta = \nu_\zeta(s,a)\,\mu(a|s)$, where ν is redefined as $\nu_\zeta(s,a) \stackrel{\text{def}}{=\!=} \zeta \min\left(\frac{1}{\pi(a|s)}, \frac{1}{\mu(a|s)}\right)$. Here, we treat $1/0 = \infty$ and $0 \cdot \infty = 0$. Then we can directly use the results of ABQ(ζ) to derive its asymptotic solution:

$$\boldsymbol{\theta}_\infty^\zeta \stackrel{\text{def}}{=\!=} \mathbf{A}_\zeta^{-1}\mathbf{b}_\zeta, \tag{8.78}$$

$$\mathbf{A}_\zeta \stackrel{\text{def}}{=\!=} X^\top \mathbf{D}_\mu \left(\mathbf{I} - \gamma \mathbf{P}_\pi \mathbf{\Lambda}_\zeta\right)^{-1} \left(\mathbf{I} - \gamma \mathbf{P}_\pi\right) X, \tag{8.79}$$

$$\mathbf{b}_\zeta \stackrel{\text{def}}{=\!=} X^\top \mathbf{D}_\mu \left(\mathbf{I} - \gamma \mathbf{P}_\pi \mathbf{\Lambda}_\zeta\right)^{-1} \mathbf{r}, \tag{8.80}$$

where the diagonal elements of $\mathbf{\Lambda}_\zeta$ are $\lambda_\zeta(s,a) = \nu_\zeta(s,a)\,\mu(a|s) = \zeta \min\left(\frac{1}{\pi(a|s)}, \frac{1}{\mu(a|s)}\right)\mu(a|s)$ for different state-action pairs.

A stable forward-view update with gradient correction corresponding to the above asymptotic solution can be derived by directly using the derivation of ABQ(ζ). The resulting algorithmn, AB-Trace(ζ), is given by the following updates:

$$\delta_t \stackrel{\text{def}}{=\!=} \mathbf{R}_{t+1} + \gamma \boldsymbol{\theta}_t^\top \bar{\boldsymbol{\phi}}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t, \tag{8.81}$$

$$\nu_\zeta(s,a) \stackrel{\text{def}}{=\!=} \zeta \min\left(\frac{1}{\pi(a|s)}, \frac{1}{\mu(a|s)}\right), \tag{8.82}$$

$$\tilde{\boldsymbol{\phi}}_{t+1} \stackrel{\text{def}}{=\!=} \sum_a \nu_\zeta(S_{t+1}, a)\pi(a|S_{t+1})\boldsymbol{\phi}(S_{t+1}, a), \tag{8.83}$$

$$\mathbf{e}_t \stackrel{\text{def}}{=\!=} \gamma \nu_{\zeta,t} \pi_t \mathbf{e}_{t-1} + \boldsymbol{\phi}_t, \tag{8.84}$$

$$\boldsymbol{\theta}_{t+1} \stackrel{\text{def}}{=\!=} \boldsymbol{\theta}_t + \alpha_t \left(\delta_t \mathbf{e}_t - \gamma \mathbf{e}_t^\top \mathbf{h}_t \left(\bar{\boldsymbol{\phi}}_{t+1} - \tilde{\boldsymbol{\phi}}_{t+1}\right)\right), \tag{8.85}$$

$$\mathbf{h}_{t+1} \stackrel{\text{def}}{=\!=} \mathbf{h}_t + \beta_t \left(\delta_t \mathbf{e}_t - \mathbf{h}_t^\top \boldsymbol{\phi}_t \boldsymbol{\phi}_t\right), \tag{8.86}$$

Note that, the factor $\nu_{\zeta,t}\pi_t$ in the eligibility trace vector update can be rewritten as:

$$\nu_{\zeta,t}\pi_t = \zeta \min\left(\frac{1}{\pi_t}, \frac{1}{\mu_t}\right)\pi_t \tag{8.87}$$

$$= \zeta \min(1, \rho_t). \tag{8.88}$$

From here, it is easy to see that, if the feature representation is tabular and the gradient correction term is dropped, then the AB-Trace algorithm reduces to the Retrace algorithm.

Finally, we remark that the action-dependent bootstrapping framework provides a principled way of developing stable and efficient off-policy algorithms as well as unifying the existing ones, where AB-Trace and its connection to Retrace is only one instance.

8.8 Conclusions

In this chapter, we have introduced the first model-free off-policy algorithm ABQ(ζ) that can produce multi-step function approximation solutions without requiring an explicit presence of importance-sampling ratios. The key to this algorithm is allowing the amount of bootstrapping to vary in an action-dependent manner, instead of keeping them constant or varying only with states. Part of this action-dependent bootstrapping factor mitigates the importance-sampling ratios while the rest of the factor is spent achieving multi-step bootstrapping. The resulting effect is that the large variance issue with importance-sampling ratios is readily removed without giving up multi-step learning. This makes ABQ(ζ) more suitable for practical use. Action-dependent bootstrapping provides an insightful and well-founded framework for deriving off-policy algorithms without importance-sampling ratios.

Chapter 9

Instability of Temporal-Difference Learning Algorithms[1]

One of the most notorious problems with off-policy learning is that a straightforward adoption of an off-policy method to parametric function approximation may diverge no matter how the step size parameter is chosen. This is a separate problem than the problem of large variance because techniques to reduce the variance do not necessarily avoid divergence.

This chapter provides insights into how some algorithms may diverge by introducing a unified approach of analyzing the stability of stochastic approximation algorithms. We investigate the problem of instability with function approximation in this unified approach and demonstrate examples of divergence of temporal-different learning algorithms in different cases above and beyond off-policy learning. This unified approach simplifies and clarifies our understanding of the instability issue of TD algorithms and constitutes our first core contribution toward the issue of instability.

9.1 Convergence of Expected Updates

Whether or not a stochastic update diverges is closely related to the choice of the step-size parameter as well as whether the deterministic update over the stationary distribution corresponding to the stochastic update converges. Therefore, a convenient way of analyzing the non-divergence or stability of a stochastic update is by investigating whether the deterministic update converges. The analysis of the corresponding deterministic update or the existence of solution of the associated "mean ODE" is also core to many convergence analysis techniques for stochastic updates, often known as *mean ODE-based proofs* (Kushner & Yin 2003). The deterministic update corresponding to a stochastic update can be derived by taking expectation of the stochastic updates with respect to the stationary Markov chain. We call this deterministic update, *the expected update*. To elaborate, let us

[1]Some of the core concepts in this chapter are developed in a published paper coauthored by this author (Sutton, Mahmood & White 2016) and summarized in another paper (Mahmood, Yu, White & Sutton 2015).

consider the following stochastic update algorithm:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(\mathbf{b}_t - \mathbf{A}_t \boldsymbol{\theta}_t \right), \tag{9.1}$$

where $\alpha > 0$, and $\mathbf{b}_t \in \mathbb{R}^n$ and $\mathbf{A}_t \in \mathbb{R}^{n \times n}$ are not functions of $\boldsymbol{\theta}_t$. Then the expected update algorithm with respect to the stationary Markov chain corresponding to the above algorithm can then be given as follows:

$$\bar{\boldsymbol{\theta}}_{t+1} = \bar{\boldsymbol{\theta}}_t + \alpha \left(\mathbf{b} - \mathbf{A} \bar{\boldsymbol{\theta}} \right), \tag{9.2}$$

where $\mathbf{b} = \mathrm{E}_0[\mathbf{b}_t]$, $\mathbf{A} = \mathrm{E}_0[\mathbf{A}_t]$, and E_0 denotes expectation with respect to the stationary Markov chain. This expected update can be rewritten as:

$$\bar{\boldsymbol{\theta}}_{t+1} = \left(\mathbf{I} - \alpha \mathbf{A} \right) \bar{\boldsymbol{\theta}}_t + \alpha \mathbf{b}. \tag{9.3}$$

The convergence of this update crucially depends on the matrix $\mathbf{I} - \alpha \mathbf{A}$. Due to its importance, we give it a separate name, *the iteration matrix*. However, as the matrix $\mathbf{A}$ is the key part of the iteration matrix, we also give it a separate name, *the key matrix*.

This update is an iterative method for solving the linear system of equations: $\mathbf{A}\bar{\boldsymbol{\theta}} = b$. A unique solution to this system exists and is given by $\mathbf{A}^{-1}\mathbf{b}$ if the key matrix $\mathbf{A}$ is non-singular. Moreover, it can be easily shown that the expected update given above converges to this solution if and only if

$$\lim_{m \to \infty} \left(\mathbf{I} - \alpha \mathbf{A} \right)^m = \mathbf{0}. \tag{9.4}$$

Whenever this condition is true, we say that the expected update and the iteration matrix are *convergent*. An equivalent condition is that the spectral radius of the iteration matrix is less than one (Varga 1962):

$$\sigma(\mathbf{I} - \alpha \mathbf{A}) < 1. \tag{9.5}$$

Here, the spectral radius of any matrix $\mathbf{B}$ is defined as

$$\sigma(\mathbf{B}) = \max_i |\operatorname{eig}_i(\mathbf{B})|, \tag{9.6}$$

with $\operatorname{eig}_i(\mathbf{B})$ being the ith eigenvalue of matrix $\mathbf{B}$. Note that the iteration matrix may not be symmetric, and therefore its eigenvalues can be complex numbers.

In the following, we provide a necessary and sufficient condition for the convergence of the iteration matrix in term of the key matrix and the step size.

Lemma 13 (Condition on key matrix and step size for convergence)**.** *The matrix* $\mathbf{I} - \alpha \mathbf{A}$ *is convergent if and only if the following condition holds:*

$$\operatorname{Re}(\operatorname{eig}_i(\mathbf{A})) > 0, \ 1 \leq i \leq n, \ \textit{and} \ \alpha < \min_j \frac{2 \operatorname{Re}(\operatorname{eig}_j(\mathbf{A}))}{\left| \operatorname{eig}_j(\mathbf{A}) \right|^2}. \tag{9.7}$$

Proof. First, we show that (9.7) is a sufficient condition. The absolute value of the ith eigenvalue of the matrix $\mathbf{I} - \alpha\mathbf{A}$ can be written as:

$$|\mathrm{eig}_i(\mathbf{I} - \alpha\mathbf{A})| = \sqrt{(1 - \alpha\,\mathrm{Re}(\mathrm{eig}_i(\mathbf{A})))^2 + (\alpha\,\mathrm{Im}(\mathrm{eig}_i(\mathbf{A})))^2} \tag{9.8}$$

$$= \sqrt{1 + \alpha^2\,|\mathrm{eig}_i(\mathbf{A})|^2 - 2\alpha\,\mathrm{Re}(\mathrm{eig}_i(\mathbf{A}))} \tag{9.9}$$

$$= \sqrt{1 + \alpha\,|\mathrm{eig}_i(\mathbf{A})|^2 \left(\alpha - \frac{2\,\mathrm{Re}(\mathrm{eig}_i(\mathbf{A}))}{|\mathrm{eig}_i(\mathbf{A})|^2}\right)} \tag{9.10}$$

$$< \sqrt{1 + \alpha\,|\mathrm{eig}_i(\mathbf{A})|^2 \left(\min_j \frac{2\,\mathrm{Re}(\mathrm{eig}_j(\mathbf{A}))}{|\mathrm{eig}_j(A)|^2} - \frac{2\,\mathrm{Re}(\mathrm{eig}_i(\mathbf{A}))}{|\mathrm{eig}_i(\mathbf{A})|^2}\right)} < 1. \tag{9.11}$$

Therefore, $\sigma(\mathbf{I} - \alpha\mathbf{A}) < 1$.

Second, we show that (9.7) is a necessary condition. It is easy to see that, if $\mathrm{Re}(\mathrm{eig}_i(\mathbf{A})) \leq 0, \exists i$, then $1 - \alpha\,\mathrm{Re}(\mathrm{eig}_i(\mathbf{A})) \geq 1$, and thus $|\mathrm{eig}_i(\mathbf{I} - \alpha\mathbf{A})| \geq 1$.

On the other hand, if $\alpha = \min_j \frac{2\,\mathrm{Re}(\mathrm{eig}_j(\mathbf{A}))}{|\mathrm{eig}_j(A)|^2} + \epsilon$ with $\epsilon \geq 0$, then for $i = \arg\min_j \frac{2\,\mathrm{Re}(\mathrm{eig}_j(\mathbf{A}))}{|\mathrm{eig}_j(A)|^2}$, we can write:

$$|\mathrm{eig}_i(\mathbf{I} - \alpha\mathbf{A})| = \sqrt{1 + \alpha\,|\mathrm{eig}_i(\mathbf{A})|^2 \left(\min_j \frac{2\,\mathrm{Re}(\mathrm{eig}_j(\mathbf{A}))}{|\mathrm{eig}_j(A)|^2} + \epsilon - \frac{2\,\mathrm{Re}(\mathrm{eig}_i(\mathbf{A}))}{|\mathrm{eig}_i(\mathbf{A})|^2}\right)} \geq 1, \tag{9.12}$$

making the matrix $\mathbf{I} - \alpha\mathbf{A}$ non-convergent. □

In the case of gradient descent updates, the iteration and the key matrices are symmetric. Therefore their eigenvalues do not have an imaginary part. In that case, Lemma 13 takes a more familiar form, which is given by the following corollary.

Corollary 2. *If the matrix* $\mathbf{A}$ *is symmetric, then* $\mathbf{I} - \alpha\mathbf{A}$ *is convergent if and only if all the eigenvalues of the matrix* $\mathbf{A}$ *are positive and the scalar parameter* α *is smaller than twice the reciprocal of the spectral radius of* $\mathbf{A}$*:*

$$\mathrm{eig}_i(\mathbf{A}) > 0,\ 1 \leq i \leq n,\ \textit{and}\ \alpha < \frac{2}{\sigma(\mathbf{A})}. \tag{9.13}$$

The step-size parameter is usually subject to tuning. Therefore, having the right key matrix $\mathbf{A}$ is key here for ensuring convergence of the expected update. The following two corollaries emphasize this important role of the key matrix.

Corollary 3. *Having all positive real parts of the eigenvalues of the matrix* $\mathbf{A}$ *is equivalent to having the matrix* $\mathbf{I} - \alpha\mathbf{A}$ *convergent, for some* $\alpha > 0$*:*

$$\mathrm{Re}\,(\mathrm{eig}_i(\mathbf{A})) > 0, 1 \leq i \leq n,\ \iff\ \mathbf{I} - \alpha\mathbf{A}\ \textit{is convergent},\ \exists \alpha > 0 \tag{9.14}$$

Therefore, having positive real parts of eigenvalues of the matrix $\mathbf{A}$ is the most important thing regarding the convergence of the expected update. There is an easy way of checking whether the eigenvalues of a matrix have positive real parts. It is to check whether a matrix is strictly diagonally dominant, which we defined below:

Definition 1 (Strictly diagonally dominant)**.** *An $n \times n$ complex matrix $\mathbf{B}$ is strictly diagonally dominant if*

$$|[\mathbf{B}]_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |[\mathbf{B}]_{ij}|, 1 \leq i \leq n. \tag{9.15}$$

Then we can use the following lemma given by Varga (1962) to check whether the eigenvalues of a matrix have positive real parts:

Lemma 14 (Eigenvalues and strictly diagonal dominance)**.** *If an $n \times n$ complex matrix $\mathbf{B}$ is strictly diagonally dominant and its diagonal entries are positive, then the eigenvalues have positive real parts:*

$$\mathrm{Re}\left(\mathrm{eig}_i(\mathbf{B})\right) > 0, \quad 1 \leq i \leq n. \tag{9.16}$$

9.2 Stability of Stochastic Updates

The stability of a stochastic update is directly related to the convergence of the expected update. Divergence of a stochastic update algorithm for any positive step size occurs when the expected update also diverges. We define a stochastic update to be *stable* if and only if the corresponding expected update is convergent. We also call a stochastic update to be *unstable* when the corresponding expected update diverges. Note that there can be cases where the expected update is not convergent but does not diverge either. For example, in the case where there are more than one solutions and the asymptotic estimate by the expected update depends on the initial $\bar{\boldsymbol{\theta}}_0$. This case is often referred to as semi-convergence (Meyer & Plemmons 1977), and we refer to the corresponding stochastic update as semi-stable.

Now, we turn into a specific algorithm based on stochastic update and analyze its corresponding expected update. For this, we choose the prototypical temporal-difference learning algorithm TD(λ). We use this algorithm to demonstrate how the key matrix $\mathbf{A}$ influences the behavior of the updates. The update of off-policy TD(λ) for state-value estimation is given as follows:

$$\mathbf{e}_t = \rho_t \left(\mathbf{e}_{t-1}\gamma_t\lambda_t + \boldsymbol{\phi}_t\right), \tag{9.17}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(R_{t+1} + \gamma_{t+1}\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) \mathbf{e}_t. \tag{9.18}$$

In order to derive the expected update of (9.18), we rewrite it as follows:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(R_{t+1} + \gamma_{t+1}\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) \mathbf{e}_t \tag{9.19}$$

$$= \boldsymbol{\theta}_t + \alpha \left(\underbrace{R_{t+1}\mathbf{e}_t}_{\mathbf{b}_t} - \underbrace{\mathbf{e}_t \left(\boldsymbol{\phi}_t - \gamma_{t+1}\boldsymbol{\phi}_{t+1}\right)^\top}_{\mathbf{A}_t} \boldsymbol{\theta}_t \right) \tag{9.20}$$

$$= \boldsymbol{\theta}_t + \alpha \left(\mathbf{b}_t - \mathbf{A}_t \boldsymbol{\theta}_t\right). \tag{9.21}$$

Then the expected update of (9.18) is given by the expectation of $\mathbf{b}_t - \mathbf{A}_t\boldsymbol{\theta}_t$ with respect to the stationary Markov chain $\{(S_t, A_t, R_{t+1})\}$ induced by the behavior policy μ. Let E_0 denote the expectation with respect to this stationary Markov chain. This follows similar steps we have taken to derive the expected update of ABQ. Therefore, the corresponding expected update of TD(λ) is given as follows:

$$\bar{\boldsymbol{\theta}}_{t+1} = \bar{\boldsymbol{\theta}}_t + \alpha \left(\mathbf{b} - \mathbf{A}\bar{\boldsymbol{\theta}}_t\right), \tag{9.22}$$

$$\mathbf{b} = \boldsymbol{\Phi}^\top \mathbf{D}_\mu \left(\mathbf{I} - \mathbf{P}_\pi \boldsymbol{\Gamma}\boldsymbol{\Lambda}\right)^{-1} \mathbf{r}_\pi, \tag{9.23}$$

$$\mathbf{A} = \boldsymbol{\Phi}^\top \mathbf{K} \boldsymbol{\Phi}, \tag{9.24}$$

$$\mathbf{K} = \mathbf{D}_\mu \left(\mathbf{I} - \mathbf{P}_\pi \boldsymbol{\Gamma}\boldsymbol{\Lambda}\right)^{-1} \left(\mathbf{I} - \mathbf{P}_\pi \boldsymbol{\Gamma}\right). \tag{9.25}$$

The stability of TD(λ) depends on whether or not the eigenvalues of its key matrix $\mathbf{A}$ has all positive real parts. Given that typically $N \gg n$, this key matrix consists of a larger $N \times N$ matrix $\mathbf{K}$ wrapped around by $\Phi^\top$ and Φ. Often we have more control over the policies and state distributions than we have over the choice of the features. Accordingly, the larger matrix $\mathbf{K}$ plays a key role in the stability of the algorithm. We call $\mathbf{K}$ the *big key matrix*.

It is not always the case that $N \geq n$, for example, in Baird's counterexample. This causes the feature matrix Φ to be column rank deficient. Furthermore, the feature matrix can be column rank deficient even when $N \geq n$. The column rank deficiency of Φ causes $\mathbf{A} = \Phi^\top \mathbf{K} \Phi$ to have eigenvalues with zero real parts, in which case the algorithm can at best be semi-stable, resulting in convergence to non-unique $\bar{\boldsymbol{\theta}}_\infty$. Although it is not practically an issue, because the corresponding approximate value function $\Phi\bar{\boldsymbol{\theta}}_\infty$ can still be unique, we assume that the feature matrix is full column rank for the convenience of analysis. Another assumption we make here is that $\mathbf{D}_\mu$ is also full rank, that is, all diagonal elements are positive.

One interesting property of the big key matrix for TD(λ) is that it always has positive real parts in all its eigenvalues, even in the off-policy case and for arbitrary values for γ_t and λ_t. Hence, it follows that in the tabular case, that is, with $\Phi = \mathbf{I}$, TD(λ) is stable. We formally describe it in the following theorem.

Theorem 31 (Positive real parts of big key matrix)**.** *For $\gamma(s), \lambda(s) \in [0, 1]$, and $d_\mu(s) > 0$, for all s, the big key matrix $\mathbf{K}$ of TD(λ) has positive real parts in all its eigenvalues:*

$$\mathrm{Re}\left(\mathrm{eig}_i(\mathbf{K})\right) > 0, \forall i. \tag{9.26}$$

Proof. In this proof, we use Lemma 14 and show that $\mathbf{K}$ is strictly diagonally dominant. In order to see that $\mathbf{K}$ has positive diagonal entries, we rewrite it as follows:

$$\begin{aligned}
\mathbf{K} &= \mathbf{D}_\mu \left(\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}\right)^{-1} \left(\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma}\right) && (9.27)\\
&= \mathbf{D}_\mu \left(\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}\right)^{-1} \left(\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda} + \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda} - \mathbf{P}_\pi \mathbf{\Gamma}\right) && (9.28)\\
&= \mathbf{D}_\mu \left[\mathbf{I} - \left(\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}\right)^{-1} \mathbf{P}_\pi \mathbf{\Gamma} \left(\mathbf{\Lambda} - \mathbf{I}\right)\right] && (9.29)\\
&= \mathbf{D}_\mu \left[\mathbf{I} - \left(\underbrace{\mathbf{I} + \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda} \mathbf{P}_\pi \mathbf{\Gamma} (\mathbf{\Lambda} - \mathbf{I}) + \left(\mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}\right)^2 \mathbf{P}_\pi \mathbf{\Gamma} (\mathbf{\Lambda} - \mathbf{I}) + \cdots}_{\mathbf{P}_\pi^\lambda}\right)\right] && (9.30)\\
&= \mathbf{D}_\mu \left(\mathbf{I} - \mathbf{P}_\pi^\lambda\right). && (9.31)
\end{aligned}$$

It is evident that the matrix $\left(\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}\right)^{-1}$ is sub-stochastic, as it is a discounted power series of the stochastic matrix $\mathbf{P}_\pi$. Therefore, $\mathbf{P}_\pi^\lambda = \left(\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}\right)^{-1} \mathbf{P}_\pi \mathbf{\Gamma} \left(\mathbf{\Lambda} - \mathbf{I}\right)$ is also a sub-stochastic matrix. Therefore, the matrices $\mathbf{I} - \mathbf{P}_\pi^\lambda$ has positive diagonal entries. For this, $\mathbf{D}_\mu \left(\mathbf{I} - \mathbf{P}_\pi^\lambda\right)$ also has positive diagonal entries.

Note that $\mathbf{D}_\mu(\mathbf{I} - \mathbf{P}_\pi^\lambda)$ also has non-positive non-diagonal entries. Strict-diagonal dominance for a matrix with positive diagonal entries and non-positive non-diagonal entries can be achieved if its row summations are positive, which is also true due to $\mathbf{P}_\pi^\lambda$ being a sub-stochastic matrix. □

It is surprising and frustrating that, although tabular TD(λ) is convergent for all the variations of discounting, bootstrapping and off-policy-ness, TD(λ) may not be convergent when extended to function approximation. In the following, we investigate different ways TD(λ) can be unstable by showing different ways its key matrix $\mathbf{A}$ can have negative real parts in its eigenvalues.

9.3 Instability Due to Off-policy Updating

TD(λ) is well known to diverge when applied with bootstrapping, off-policy updating, and function approximation. This amounts to $\exists s, \lambda(s) < 1$, $\pi \neq \mu$, and $N > n$. To demonstrate this, we consider the following continuing off-policy policy-evaluation task. In this task, there are two states, $\mathcal{S} = \{1, 2\}$ and only a single feature, hence: $N > n$. From each state, there are two actions available: $\mathcal{A} = \{1, 2\}$. Let $\mathbf{Q} \in \mathbb{R}^{|\mathcal{S}| \cdot |\mathcal{A}| \times |\mathcal{S}|}$ denote the transition probability for each given state-action pair: $[\mathbf{Q}]_{sa,:} = p(\cdot|s, a)$. The transition probability for each state given state-action pairs is as follows:

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{9.32}$$

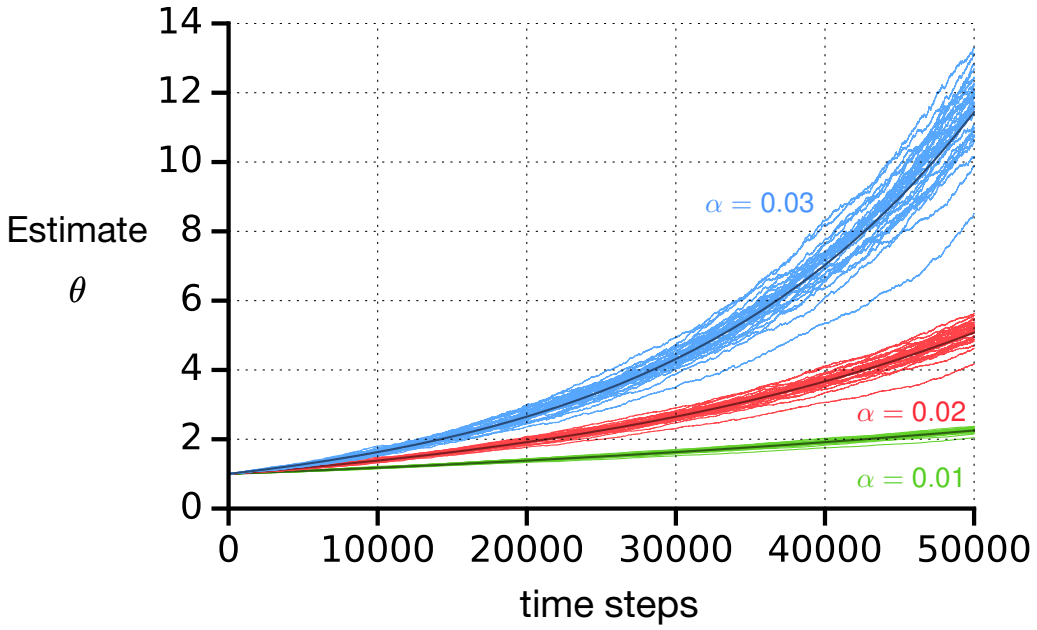

Figure 9.1: Demonstration of instability of TD(λ) on an off-policy prediction task.

The feature matrix is given by $\Phi = [[0.5], [1]]^\top$. The behavior policy is given by $\boldsymbol{\mu} = [0.5, 0.5, 0.5, 0.5]$, and the target policy is given by $\boldsymbol{\pi} = [0.1, 0.9, 0.1, 0.9]$. The discount factor and the bootstrapping parameter are state independent: $\gamma = 0.9$, and $\lambda = 0$. Then the state-distribution induced by the behavior policy is $\mathbf{d}_\mu = [0.5, 0.5]^\top$. The state-to-state transition probability matrix induced by the target policy is given by $[\mathbf{P}_\pi]_{ss'} = [\boldsymbol{\mu}]_s [\mathbf{Q}]_{s:,s'}$:

$$\mathbf{P}_\pi = \begin{bmatrix} 0.1 & 0.9 \\ 0.1 & 0.9 \end{bmatrix}. \tag{9.33}$$

Reward is zero everywhere.

The resulting big key matrix $\mathbf{K}$ has the following eigenvalues: $0.5, 0.05$. And the resulting key matrix $\mathbf{A}$ is given by $\mathbf{A} = \Phi^\top \mathbf{K} \Phi = -0.01625 < 0$. It indicates that the off-policy TD(λ) algorithm in this task is not stable.

To illustrate this instability empirically, we run the stochastic updates of TD(λ) on this task, with the initial estimate $\boldsymbol{\theta}_0 = 1$. Figure 9.1 illustrates the instability of off-policy TD(λ) on this task for three different step sizes $[0.01, 0.02, 0.03]$. The plot shows the estimated parameter $\boldsymbol{\theta}$ over time. The dark curves correspond to the expected update, and the bright curves correspond to the stochastic updates, 30 of them for each step size generated using independent trajectories. Although the true solution is zero, all the estimates diverge away from zero with time.

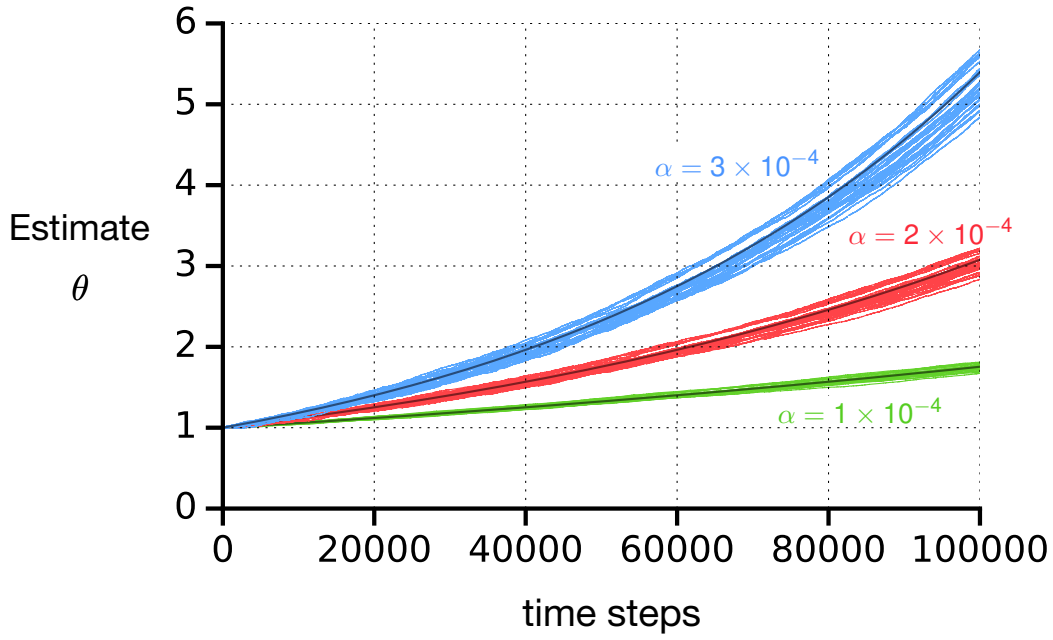

Figure 9.2: Demonstration of instability of TD(λ) on an on-policy prediction task with state-dependent bootstrapping.

9.4 Instability Due to State-Dependent Bootstrapping

Now, we show that with function approximation, even on-policy TD(λ) can be unstable. This may happen when the bootstrapping parameter λ is state-dependent. For that, consider that the two-state MDP in the previous section (9.3). This time, the target policy is the same as the behavior policy: $\boldsymbol{\pi} = \boldsymbol{\mu} = [0.5, 0.5, 0.5, 0.5]$. Then the transition probability matrix induced by the target policy is given by the following:

$$\mathbf{P}_\pi = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}, \tag{9.34}$$

and the state-distribution induced by the target policy is $\mathbf{d}_\pi = [0.5, 0.5]^\top$. The discount factor is $\gamma = 0.99$, and the bootstrapping parameter is state-dependent: $\lambda(1) = 1.0, \lambda(2) = 0.8$. The rest remain the same. Then the resulting key matrix $\mathbf{A}$ is given by $\mathbf{A} = \Phi^\top \mathbf{K} \Phi \approx -0.056 < 0$.

Figure 9.2 shows the estimates by stochastic update of TD(λ) for step sizes $[1, 2, 3] \times 10^{-4}$. As before, the estimates diverges away from zero with time.

9.5 Instability Due to Selective Updating

With function approximation, on-policy TD(λ) may diverge even with a single global λ. This happens when updates are scaled selectively according to a user-defined interest of the

states. Let us define the user-defined state-dependent interest function as $i : \mathcal{S} \to (0, \infty)$. We use the shorthand $I_t = i(S_t)$. Then such user-defined interest can be incorporated into TD(λ) by scaling the forward-view update of TD(λ) for state S_t with I_t:

$$\Delta\boldsymbol{\theta}_t = \alpha I_t \left(G_t^\lambda - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) \boldsymbol{\phi}_t. \tag{9.35}$$

The equivalent backward-view update of TD(λ) then is given by the following:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(R_{t+1} + \gamma_{t+1}\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t\right) \mathbf{e}_t, \tag{9.36}$$

$$\mathbf{e}_t = \left(\mathbf{e}_{t-1}\gamma_t\lambda_t + I_t\boldsymbol{\phi}_t\right) \rho_t, \tag{9.37}$$

where the interest function appears in the update of the eligibility trace vector. This is a generalization of the standard off-policy TD(λ) update, which can be obtained by having $I_t = 1, \forall t$.

It is easy to see that the corresponding expected update is as follows:

$$\bar{\boldsymbol{\theta}}_{t+1} = \bar{\boldsymbol{\theta}}_t + \alpha \left(\mathbf{b} - \mathbf{A}\bar{\boldsymbol{\theta}}_t\right), \tag{9.38}$$

$$\mathbf{b} = \Phi^\top \mathbf{D}_{\mu\cdot i} \left(\mathbf{I} - \mathbf{P}_\pi \boldsymbol{\Gamma}\boldsymbol{\Lambda}\right)^{-1} \mathbf{r}_\pi, \tag{9.39}$$

$$\mathbf{A} = \Phi^\top \mathbf{K} \Phi, \tag{9.40}$$

$$\mathbf{K} = \mathbf{D}_{\mu\cdot i} \left(\mathbf{I} - \mathbf{P}_\pi \boldsymbol{\Gamma}\boldsymbol{\Lambda}\right)^{-1} \left(\mathbf{I} - \mathbf{P}_\pi \boldsymbol{\Gamma}\right). \tag{9.41}$$

Here, the diagonal matrix $\mathbf{D}_{\mu\cdot i}$ contains the product of the state-distribution and interest in its diagonal elements: $[\mathbf{D}_{\mu\cdot i}]_{s,s} = [\mathbf{d}_\mu]_s i(s)$.

Now, we provide an on-policy policy evaluation task where selective updating causes instability. For that, we use the same two-state MDP from Section 9.3. Here, the target policy is the same as the behavior policy: $\pi = [0.5, 0.5]^\top$. The discount factor is $\gamma = 0.99$. The bootstrapping parameter is state-independent: $\lambda(s) = 0.5, \forall s$. The state-dependent interest function is given by $i(1) = 1.6, i(2) = 0.4$. Then the resulting key matrix is $\mathbf{A} \approx -0.041 < 0$.

Figure shows the instability of the stochastic update of TD(λ) in this task for step sizes $[1, 2, 3] \times 10^{-3}$.

9.6 Stability with Arbitrary State-Dependent Discounting

It is reasonable to wonder whether on-policy TD(λ) with function approximation is unstable for state-dependent discounting. The suspicion is more relevant because we found on-policy TD(λ) to be unstable in two different ways involving state-dependent functions. Surprisingly, it is not the case with state-dependent discounting. It is because, in this case, the big key matrix $\mathbf{K}$ not only has positive real parts in all its eigenvalues, but also is positive definite under the usual assumptions. It guarantees that the key matrix $\mathbf{A}$ has positive real parts in all its eigenvalues. The following lemma describes this.

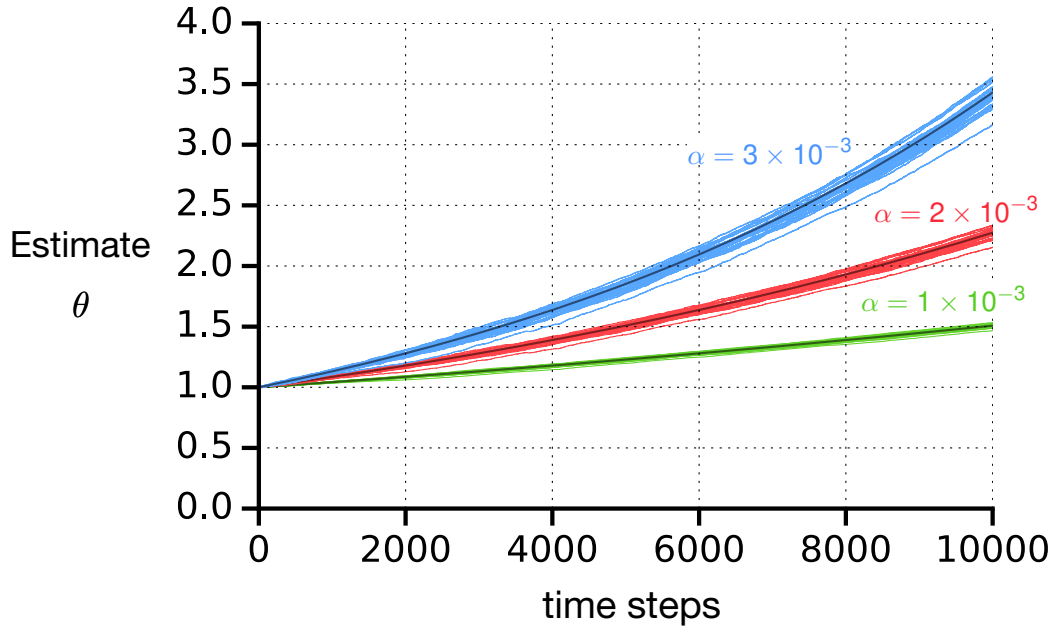

Figure 9.3: Demonstration of instability of TD(λ) on an on-policy prediction task with selective updating.

Lemma 15 (Sufficient condition for positive-definite key matrix)**.** *Let us consider an asymmetric positive definite matrix* $\mathbf{K} \in \mathbb{R}^{N \times N}$*. Then* $\mathbf{A} = \Phi^\top \mathbf{K} \Phi$ *is positive definite, for any full column rank matrix* $\Phi \in \mathbb{R}^{N \times n}$*.*

Proof. $\mathbf{K}$ is positive definite if and only if $\boldsymbol{\phi}^\top \mathbf{K} \boldsymbol{\phi} > 0$, for $\boldsymbol{\phi} \neq \mathbf{0}$. If Φ is full column rank, then $\mathbf{y} = \Phi \boldsymbol{\phi} \neq \mathbf{0}$, for $\boldsymbol{\phi} \neq \mathbf{0}$. Therefore, $\mathbf{y}^\top \mathbf{K} \mathbf{y} = \boldsymbol{\phi}^\top \Phi^\top \mathbf{K} \Phi \boldsymbol{\phi} = \boldsymbol{\phi}^\top \mathbf{A} \boldsymbol{\phi} > 0$, for $\boldsymbol{\phi} \neq \mathbf{0}$, from which it follows that $\mathbf{A}$ is positive definite. □

Having positive real parts of all eigenvalues follows from the fact that $\mathbf{A}$ is positive definite. However, the converse does not hold for asymmetric matrices. We use the following lemma involving asymmetric positive definite matrices.

Lemma 16 (Positive-definiteness of asymmetric matrices)**.** *For an asymmetric square real matrix* $\mathbf{K}$*, the following statements are equivalent:*

1. $\mathbf{K}$ *is positive definite.*
2. $\mathbf{K} + \mathbf{K}^\top$ *is positive definite.*
3. $\mathrm{Re}(\mathrm{eig}_i(\mathbf{K} + \mathbf{K}^\top)) > 0, \forall i.$

Yet another result we use is as follows:

Lemma 17. $\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}$ *is an* M*-matrix.*

Proof. A matrix is a Z-matrix if its off-diagonal elements are non-positive. A matrix is a M-matrix if it is a Z-matrix and can be written in the form $s\mathbf{I} - \mathbf{B}$, where $\mathbf{B}$ has only non-negative entries and $s > \sigma(\mathbf{B})$. Accordingly, $\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}$ is a Z-matrix. On the other hand, the spectral radius of $\mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}$ is less than one: $\sigma(\mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}) < 1$, according to Perron–Frobenius theorem. Therefore, according to the definition of M-matrix, $\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}$ is an M-matrix. □

As $\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda}$ is an M-matrix, its inverse has all non-negative entries.

In the following, we show that when the user-defined interest is uniform $i(s) = 1, \forall s$ and λ is state-independent, on-policy TD(λ) is stable even with state-dependent discounting and linear function approximation.

Theorem 32 (Stability with state-dependent discounting)**.** *With state-dependent discounting* $\gamma(s) \in [0, 1), \forall s$*, a single global* $\lambda \in [0, 1]$*, that is,* $\mathbf{\Lambda} = \lambda \mathbf{I}$*, and* $\mu = \pi$*, the big key matrix* $\mathbf{K} = \mathbf{D}_\pi (\mathbf{I} - \mathbf{P}_\pi^\lambda)$ *of TD*(λ) *is positive definite.*

Proof. We show that $\mathbf{K} + \mathbf{K}^\top$ is strictly diagonally dominant, and with positive diagonal entries, from which it follows that $\text{Re}(\text{eig}_i(\mathbf{K} + \mathbf{K}^\top)) > 0, \forall i$ by Lemma 14.

We already showed that $\mathbf{K}$ has positive diagonal entries, which leads to the fact $\mathbf{K} + \mathbf{K}^\top$ has positive diagonal entries. For $\mathbf{K} + \mathbf{K}^\top$ to be strictly diagonally dominant, it is enough to show that its row sums and column sums are positive. We already showed that its row sums are positive. The sub-stochastic matrix $\mathbf{P}_\pi^\lambda$ can be re-written as follows:

$$\begin{aligned}
\lambda \mathbf{P}_\pi^\lambda &= \lambda \left[\mathbf{I} - (\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} (\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma}) \right] && (9.42) \\
&= \lambda \left[\mathbf{I} - (\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} (\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma} + \lambda \mathbf{P}_\pi \mathbf{\Gamma} - \mathbf{P}_\pi \mathbf{\Gamma}) \right] && (9.43) \\
&= (1 - \lambda)(\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} \lambda \mathbf{P}_\pi \mathbf{\Gamma} && (9.44) \\
&= (1 - \lambda)(\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} (\lambda \mathbf{P}_\pi \mathbf{\Gamma} - \mathbf{I} + \mathbf{I}) && (9.45) \\
&= (1 - \lambda) \left((\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} - \mathbf{I} \right). && (9.46)
\end{aligned}$$

The column sum can be written as $\mathbf{1}^\top \mathbf{K}$. Therefore, we can write:

$$\begin{aligned}
\lambda \mathbf{1}^\top \mathbf{K} &= \lambda \mathbf{1}^\top \mathbf{K} (\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})(\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} && (9.47) \\
&= \mathbf{1}^\top \mathbf{D}_\pi (\lambda \mathbf{I} - \lambda \mathbf{P}_\pi^\lambda)(\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})(\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} && (9.48) \\
&= \mathbf{d}_\pi^\top \left(\lambda \mathbf{I} - (1 - \lambda) \left((\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} - \mathbf{I} \right) \right) (\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})(\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} && (9.49) \\
&= \mathbf{d}_\pi^\top \left(\mathbf{I} - (1 - \lambda)(\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} \right) (\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})(\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} && (9.50) \\
&= \mathbf{d}_\pi^\top \left(\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma} - (1 - \lambda) \mathbf{I} \right) (\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1} && (9.51) \\
&= \lambda \left(\mathbf{d}_\pi \circ (\mathbf{1} - \boldsymbol{\gamma}) \right)^\top (\mathbf{I} - \lambda \mathbf{P}_\pi \mathbf{\Gamma})^{-1}. && (9.52)
\end{aligned}$$

Each element of $(\mathbf{d}_\pi \circ (\mathbf{1} - \boldsymbol{\gamma}))$ is positive, and $(\mathbf{I} - \lambda \mathbf{P}_\pi \boldsymbol{\Gamma})$ is M-matrix. Therefore, $(\mathbf{I} - \lambda \mathbf{P}_\pi \boldsymbol{\Gamma})^{-1}$ has non-negative entries. Hence, $\mathbf{1}^\top \mathbf{K}$ has all positive element for $\lambda > 0$. On the other hand, when $\lambda = 0$, we can writen:

$$\mathbf{1}^\top \mathbf{K} = \mathbf{1}^\top \mathbf{D}_\pi (\mathbf{I} - \mathbf{P}_\pi \boldsymbol{\Gamma}) \tag{9.53}$$

$$= \mathbf{d}_\pi^\top (\mathbf{I} - \mathbf{P}_\pi \boldsymbol{\Gamma}) \tag{9.54}$$

$$= (\mathbf{d}_\pi \circ (\mathbf{1} - \boldsymbol{\gamma}))^\top, \tag{9.55}$$

where each element is positive. Therefore, column sums of $\mathbf{K}$ are also positive, from which it follows that $\mathbf{K}$ is positive definite. □

9.7 Oscillation Due to Asymmetric Iteration Matrix

Finally, we illustrate a particular characteristic of TD updates—oscillation—that is not observed in standard gradient-descent updates. It is often observed that TD estimates produce oscillatory learning curves, both when the update is stable and unstable (Sutton & Barto 1998). It is originated due to asymmetric key matrices, due to which they can have eigenvalues with a non-zero imaginary part. The real part of the eigenvalues of the key matrix, when positive, determines the rate of decay. On the other, the imaginary part of the eigenvalues determines the frequency of the oscillation, the larger the magnitude, the shorter the wavelength.

Note that the key matrix starts to have imaginary parts in its eigenvalues for $N \geq 3$. For $N = 1$, the key matrix is scalar and always real. In order to show that the key matrix has real eigenvalues for $N = 2$, we use the following lemma.

Lemma 18 (Eigenvalues of 2×2 matrix)**.** *For a 2×2 real matrix $\mathbf{B}$, the eigenvalues are given by:*

$$\lambda = \frac{-\operatorname{trace}(\mathbf{B}) \pm \sqrt{\operatorname{trace}(\mathbf{B}) - 4 \det(\mathbf{B})}}{2}. \tag{9.56}$$

The following theorem states that a 2×2 big key matrix always has real eigenvalues, from which it follows that the key matrix $\mathbf{A}$ for $N = 2$ also has all real eigenvalues.

Theorem 33 (Real eigenvalues of 2×2 matrix)**.** *For $N = 2$, the big key matrix $\mathbf{K} = \mathbf{D}_\mu (\mathbf{I} - \mathbf{P}_\pi^\lambda)$ has real eigenvalues.*

Proof. For this, we have to show that $\operatorname{trace}(\mathbf{K}) \geq 4 \det(\mathbf{K})$. We already showed in Theorem 31 that $\mathbf{K}$ has positive diagonal elements and non-positive off-diagonal elements. Say, the elements of $(\mathbf{I} - \mathbf{P}_\pi^\lambda)$ are as follows:

$$\mathbf{I} - \mathbf{P}_\pi^\lambda = \begin{bmatrix} p_1 & p_{12} \\ p_{21} & p_2 \end{bmatrix}. \tag{9.57}$$

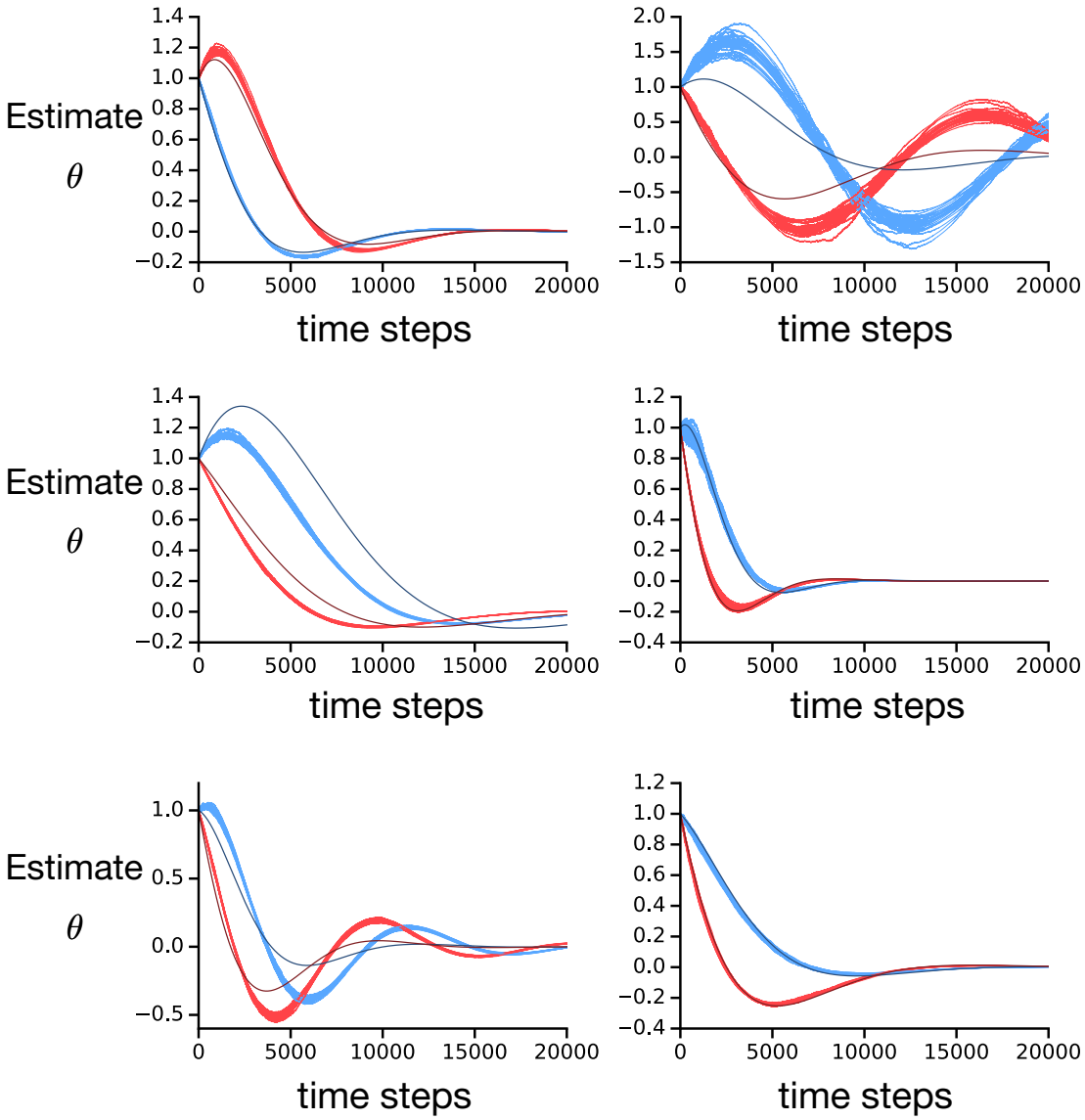

Figure 9.4: Estimates of TD(λ) may oscillate due to the asymmetry of its associated iteration matrix. This is shown in six different on-policy prediction tasks. The two curves in each plot are the two elements of the parameter vector $\boldsymbol{\theta}$.

The stationary state probabilities are $\mathbf{u}_\mu = [d_1, d_2]^\top$. Therefore, $p_1, p_2 \in (0, 1]$, and p_{12}, $p_{21} \in (-1, 0]$. Also note that, $d_1 = 1 - d_2$. Therefore, the trace and determinant of the key matrix are given by: $\text{trace}(\mathbf{K}) = d_1 p_1 + d_2 p_2 > 0$, $\det(\mathbf{K}) = d_1 d_2 p_1 p_2 - d_1 d_2 p_{12} p_{21} \leq d_1 d_2 p_1 p_2$.

Hence, it suffices to show that $d_1 p_1 + d_2 p_2 - 4 d_1 d_2 p_1 p_2 \geq 0$.

We can assume $d_1 \geq d_2$ without any loss of generality. Therefore, we can write:

$$
\begin{aligned}
d_1 p_1 + d_2 p_2 - 4 d_1 d_2 p_1 p_2 &= d_1 p_1 + (1 - d_1) p_2 - 4 d_1 (1 - d_1) p_1 p_2 && (9.58)\\
&= d_1 p_1 + (1 - d_1) p_2 - p_1 p_2 + \left(4 d_1^2 - 4 d_1 + 1\right) p_1 p_2 && (9.59)\\
&= d_1 p_1 + (1 - d_1) p_2 - p_1 p_2 + (2 d_1 - 1)^2 p_1 p_2 && (9.60)\\
&\geq d_1 p_1 + (1 - d_1) p_2 - p_1 p_2 && (9.61)\\
&= d_1 (p_1 - p_2) + p_2 (1 - p_1) && (9.62)\\
&\geq d_1 \left(p_1 - p_2 + p_2 - p_1 p_2\right); \text{ as } p_2 (1 - p_1) \geq d_1 p_2 (1 - p_1) && (9.63)\\
&= d_1 p_1 \left(1 - p_2\right) \geq 0. && (9.64)
\end{aligned}
$$

□

In order to illustrate the oscillation in TD update, we randomly generate MDPs with three states. We chose $n = 2, |\mathcal{A}| = 2$, and consider only on-policy tasks. The state-action pair to next state probabilities, behavior policy μ, the target policy π, the discount matrix $\mathbf{\Gamma}$, the bootstrapping matrix $\mathbf{\Lambda}$, and the feature matrix Φ were generated uniformly randomly. From them we chose six, for which the real parts of the eigenvalues are positive but smaller than the magnitude of the imaginary parts, so that the decay is slow enough to make the oscillation visible. In each case, we ran both the expected and 30 runs of stochastic update of TD(λ). Figure 9.4 shows the results for step size 0.1. In each case, both the expected (dark curves) and estimated $\boldsymbol{\theta}$ (bright curves) showed similar damped oscillation.

9.8 Conclusions

In this chapter, we explored the issue of instability by analyzing TD(λ) in different kinds of prediction tasks with both on-policy and off-policy scenarios. We utilized the concept of deterministic or expected updates corresponding to stochastic approximation algorithms. The key to our analysis revolved around the properties of a certain "key" matrix associated with the expected updates of the TD(λ) algorithm. TD(λ) becomes unstable whenever the key matrix of expected updates has negative eigenvalues. We utilized this fact to demonstrate the instability of TD(λ) in three different cases: off-policy updating, updates with state-dependent bootstrapping and selective updating. We have also provided a result showing that TD(λ) is stable with state-dependent discounting. Finally, we showed that TD updates may oscillated, which resulted from the asymmetry of the iteration matrix of its expected update.

Chapter 10

An Emphatic Approach to Stable Temporal-Difference Learning[1]

In this chapter, we introduce a systematic approach to ensuring the stability of off-policy temporal-difference learning algorithms, a key contribution in this thesis. In addition to being stable under off-policy updating, this approach also ensures stability under the other cases where TD(λ) can be unstable. The core idea behind this approach is ensuring positive definiteness of the "key" matrix involved in the expected update, which causes the expected update to be convergent. We show that it amounts to warping the effective stationary state distribution. The resulting algorithms can be seen as selectively emphasizing and de-emphasizing the updates. We call them *emphatic algorithms*. We show results confirming that emphatic algorithms are stable.

Sutton et al. (2009) introduced another set of stable off-policy algorithms that are derived by following a gradient-based approach to temporal-difference learning. We used this approach to ensure the stability of algorithms based on action-dependent bootstrapping in Chapter 8. However, this approach requires having two sets of parameters and adjusting two different step sizes, which substantially increases the amount of parameter search. The advantage of emphatic algorithms over gradient-based TD algorithms is that emphatic algorithms only require tuning a single scalar step-size parameter and thus are practically more convenient.

10.1 Warping the Update Distribution for Stability

Stability of temporal-difference learning algorithms can be ensured by having their big key matrix $\mathbf{K}$ to be a positive definite matrix, as we have observed in Chapter 9. We have also observed that if the matrix $\mathbf{K} + \mathbf{K}^\top$ is strictly diagonally dominant, which amounts to showing that the row and the column sums are positive, then $\mathbf{K}$ is positive definite. However, we have also seen that it is generally not true for TD(λ). Although the row

[1] The core concepts in this chapter are developed in a published paper coauthored by this author (Sutton, Mahmood & White 2016), which is summarized in another paper (Mahmood, Yu, White & Sutton 2015).

sums are always positive, the column sums may not be. To seek an opportunity to fix this problem, let us look at the big key matrix of the most general form of TD(λ):

$$\mathbf{K} = \mathbf{D}_{\mu \cdot i} \left(\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \mathbf{\Lambda} \right)^{-1} \left(\mathbf{I} - \mathbf{P}_\pi \mathbf{\Gamma} \right) = \mathbf{D}_{\mu \cdot i} \left(\underbrace{\mathbf{I} - \mathbf{P}_\pi^\lambda}_{\mathbf{B}} \right) = \mathbf{D}_{\mu \cdot i} \mathbf{B}. \tag{10.1}$$

Salvaging the updates from instability amounts to sculpting this key matrix, but at the same time being careful about preserving the core features of TD solutions. The big key matrix has two separate components. The matrix $\mathbf{B} \stackrel{\text{def}}{=\!=} \mathbf{I} - \mathbf{P}_\pi^\lambda$ determines the form of the multi-step TD error, and hence we call it the *TD-error matrix*. The matrix $\mathbf{D}_{\mu \cdot i}$ determines the distribution of updating states, which we call *the effective stationary state distribution* as the updating states can be seen being sampled according to this distribution. With uniform interest, the effective stationary state distribution equates the actual stationary state distribution induced by the behavior policy. However, as we have observed in Section 9.5, the stationary state distribution can be arbitrarily warped by scaling the updates in a state-dependent manner. User-defined interest function is one such way of achieving that.

Here, we aim at preserving the TD-error matrix $\mathbf{B}$ and warp the effective stationary state distribution to produce big key matrices that are positive definite. Let us say the effective stationary state distribution vector $\mathbf{m}$ is such that the resulting big key matrix $\mathbf{K} = \mathbf{MB}$ is positive definite, where $\mathbf{M} = \text{diag}(\mathbf{m})$. As $\mathbf{m}$ is a warping of the original stationary state distribution based on user-defined interest $\mathbf{d}_{\mu \cdot i}$, we can write $\mathbf{m} = \mathbf{W} \mathbf{d}_{\mu \cdot i}$. Where $\mathbf{W}$ is a $N \times N$ matrix, which we call *the warping matrix*. Our goal is to find a warping matrix $\mathbf{W}$ such that the resulting big key matrix $\mathbf{K}$ is positive definite, which we can achieve by having the column sums $\mathbf{1}^\top \mathbf{K}$ to be positive. The column sums with this warped stationary state distribution $\mathbf{m}$ can be written as follows:

$$\mathbf{1}^\top \mathbf{K} = \mathbf{1}^\top \mathbf{MB} \tag{10.2}$$

$$= \mathbf{m}^\top \mathbf{B} \tag{10.3}$$

$$= \mathbf{d}_{\mu \cdot i}^\top \mathbf{W}^\top \mathbf{B}. \tag{10.4}$$

One way we can ensure positive column sums is by choosing $\mathbf{W}$ in such a way that $\mathbf{W}^\top \mathbf{B}$ is a non-negative matrix.

Consider the following warping matrix:

$$\mathbf{W} = \left(\mathbf{B}^\top \right)^{-1}. \tag{10.5}$$

Then $\mathbf{W}^\top \mathbf{B}$ is a non-negative matrix, as:

$$\mathbf{W}^\top \mathbf{B} = \mathbf{I}. \tag{10.6}$$

Therefore, the resulting key matrix $\mathbf{A}$ and the corresponding expected update are stable. However, achieving stability of the expected update is not enough. We need stochastic updates based on them.

10.2 Emphatic Temporal Difference Learning Algorithms

In this section, we derive a systematic way of deriving stochastic updates based on stable expected updates with warped stationary state distribution. Here, the key observation is that, any warped stationary state distribution can be viewed as forming a particular interest function:

$$\mathbf{m} = \mathbf{d}_\mu \circ \left(\mathbf{d}_\mu^{-1} \circ \mathbf{m}\right) = \mathbf{d}_\mu \circ \mathbf{m}_0, \tag{10.7}$$

where $\mathbf{m}_0 = \mathbf{d}_\mu^{-1} \circ \mathbf{m}$ can be viewed as a particular form of interest function. We investigate this interest function further:

$$\begin{aligned}
\mathbf{m}_0 = \mathbf{d}_\mu^{-1} \circ \mathbf{m} &= \mathbf{d}_\mu^{-1} \circ (\mathbf{W}\mathbf{d}_{\mu \cdot i}) && (10.8)\\
&= \mathbf{d}_\mu^{-1} \circ (\mathbf{W}\left(\mathbf{d}_\mu \circ \mathbf{i}\right)). && (10.9)
\end{aligned}$$

As with user-defined interest, we can also have stochastic updates associated with this particular interest by applying its elements $[\mathbf{d}_\mu^{-1} \circ \mathbf{m}]_s$ for the updating states s. We can view this scaling function as a way of emphasizing or de-emphasizing the updates depending on whether $[\mathbf{d}_\mu^{-1} \circ \mathbf{m}]_s$ is greater than unity or not. Likewise, we call any approach based on sculpting the stationary-state distribution for achieving stability to be an emphatic approach.

We do not have access to the model to directly compute $\mathbf{d}_\mu^{-1} \circ \mathbf{m}$. But we can use an unbiased estimate of this term. We derive a scalar estimate of $\mathbf{d}_\mu^{-1} \circ \mathbf{m}$, for the choice of $\mathbf{W} = \left(\mathbf{B}^\top\right)^{-1}$. With this choice, we can write the emphatic vector $\mathbf{m}_0$ as

$$\begin{aligned}
\mathbf{m}_0 &= \mathbf{d}_\mu^{-1} \circ \left[\left(\mathbf{B}^\top\right)^{-1} \left(\mathbf{d}_\mu \circ \mathbf{i}\right)\right] && (10.10)\\
&= \mathbf{d}_\mu^{-1} \circ \left[\left(\mathbf{I} - \mathbf{\Gamma}\mathbf{\Lambda}\mathbf{P}_\pi^\top\right)\left(\mathbf{I} - \mathbf{\Gamma}\mathbf{P}_\pi^\top\right)^{-1} \left(\mathbf{d}_\mu \circ \mathbf{i}\right)\right] && (10.11)\\
&= \mathbf{d}_\mu^{-1} \circ \left[\left(\mathbf{\Lambda}\left(\mathbf{I} - \mathbf{\Gamma}\mathbf{P}_\pi^\top\right) + \left(\mathbf{I} - \mathbf{\Lambda}\right)\right)\left(\mathbf{I} - \mathbf{\Gamma}\mathbf{P}_\pi^\top\right)^{-1} \left(\mathbf{d}_\mu \circ \mathbf{i}\right)\right] && (10.12)\\
&= \left(\mathbf{\Lambda}\mathbf{i} + \left(\mathbf{I} - \mathbf{\Lambda}\right)\left(\underbrace{\mathbf{d}_\mu^{-1} \circ \left(\mathbf{I} - \mathbf{\Gamma}\mathbf{P}_\pi^\top\right)^{-1} \left(\mathbf{d}_\mu \circ \mathbf{i}\right)}_{\mathbf{f}}\right)\right) && (10.13)\\
&= \mathbf{\Lambda}\mathbf{i} + \left(\mathbf{I} - \mathbf{\Lambda}\right)\mathbf{f}. && (10.14)
\end{aligned}$$

The first term of the above can be sampled by using state-dependent bootstrapping and interest. In order to see how to sample the second term, let us write $\mathbf{f}$ in the following way:

$$\begin{aligned}
\mathbf{f} = \mathbf{d}_\mu^{-1} \circ \left(\mathbf{I} - \mathbf{\Gamma}\mathbf{P}_\pi^\top\right)^{-1} \left(\mathbf{d}_\mu \circ \mathbf{i}\right) && (10.15)\\
\left(\mathbf{I} - \mathbf{\Gamma}\mathbf{P}_\pi^\top\right)\left(\mathbf{f} \circ \mathbf{d}_\mu\right) = \mathbf{d}_\mu \circ \mathbf{i} && (10.16)\\
\mathbf{f} \circ \mathbf{d}_\mu = \mathbf{\Gamma}\mathbf{P}_\pi^\top \left(\mathbf{f} \circ \mathbf{d}_\mu\right) + \mathbf{d}_\mu \circ \mathbf{i} && (10.17)
\end{aligned}$$

$$\mathbf{f} = \mathbf{i} + \mathbf{d}_\mu^{-1} \circ \left(\mathbf{\Gamma} \mathbf{P}_\pi^\top \left(\mathbf{f} \circ \mathbf{d}_\mu \right) \right). \tag{10.18}$$

Therefore, each element can be written as:

$$[\mathbf{f}]_s = i(s) + \gamma(s) \sum_{\bar{s},\bar{a}} \frac{p(s|\bar{a},\bar{s})\pi(\bar{a}|\bar{s})d_\mu(\bar{s})}{d_\mu(s)} [\mathbf{f}]_{\bar{s}} \tag{10.19}$$

$$= i(s) + \gamma(s) \sum_{\bar{s},\bar{a}} \frac{p_\mu(s,\bar{a}|\bar{s})d_\mu(\bar{s})}{d_\mu(s)} \frac{\pi(\bar{a}|\bar{s})}{\mu(\bar{a}|\bar{s})} [\mathbf{f}]_{\bar{s}} \tag{10.20}$$

$$= i(s) + \gamma(s) \sum_{\bar{s},\bar{a}} p_\mu(\bar{s},\bar{a}|s) \frac{\pi(\bar{a}|\bar{s})}{\mu(\bar{a}|\bar{s})} [\mathbf{f}]_{\bar{s}}. \tag{10.21}$$

Then the following scalar statistic F_t is an unbiased estimate of $[\mathbf{f}]_{S_t}$ under the corresponding stationary Markov chain:

$$F_t = I_t + \gamma_t \rho_{t-1} F_{t-1}. \tag{10.22}$$

Similarly, the following scalar statistic M_t is an unbiased estimate of $[\mathbf{m}_o]_{S_t}$ under the corresponding stationary Markov chain:

$$M_t = \lambda_t I_t + (1 - \lambda_t) F_t. \tag{10.23}$$

We call M_t *the update emphasis* as this is used as a scaling factor for each update. Then the resulting backward-view updates can be written as:

$$\delta_t = R_{t+1} + \gamma_{t+1} \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t, \tag{10.24}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \delta \mathbf{e}_t, \tag{10.25}$$

$$\mathbf{e}_t = \rho_t \left(\mathbf{e}_{t-1} \gamma_t \lambda_t \rho_t + M_t \boldsymbol{\phi}_t \right), \tag{10.26}$$

where the update emphasis appears in the update of eligibility trace vector, similarly to the user-defined interest function I_t in Section 9.5. We call this algorithm the *emphatic temporal-different learning algorithm*, in short ETD(λ). It is evident from the above analysis that ETD(λ) is stable, which we state in the following theorem.

Theorem 34 (Stability of Emphatic TD(λ))**.** *For any*

- *Markov decision process* $\{S_t, A_t, R_{t+1}\}_{t=0}^\infty$ *with finite state and action sets* $\mathcal{S}$ *and* $\mathcal{A}$*,*
- *behavior policy* μ *with a stationary invariant distribution* $d_\mu(s) > 0, \forall s \in \mathcal{S}$*,*
- *target policy* π *with coverage, i.e., s.t., if* $\pi(a|s) > 0$*, then* $\mu(a|s) > 0$*,*
- *discount function* $\gamma : \mathcal{S} \to [0,1]$ *s.t.* $\prod_{k=1}^\infty \gamma(S_{t+k}) = 0, w.p.1, \forall t > 0$*,*
- *bootstrapping function* $\lambda : \mathcal{S} \to [0,1]$*,*
- *interest function* $i : \mathcal{S} \to (0, \infty)$*,*

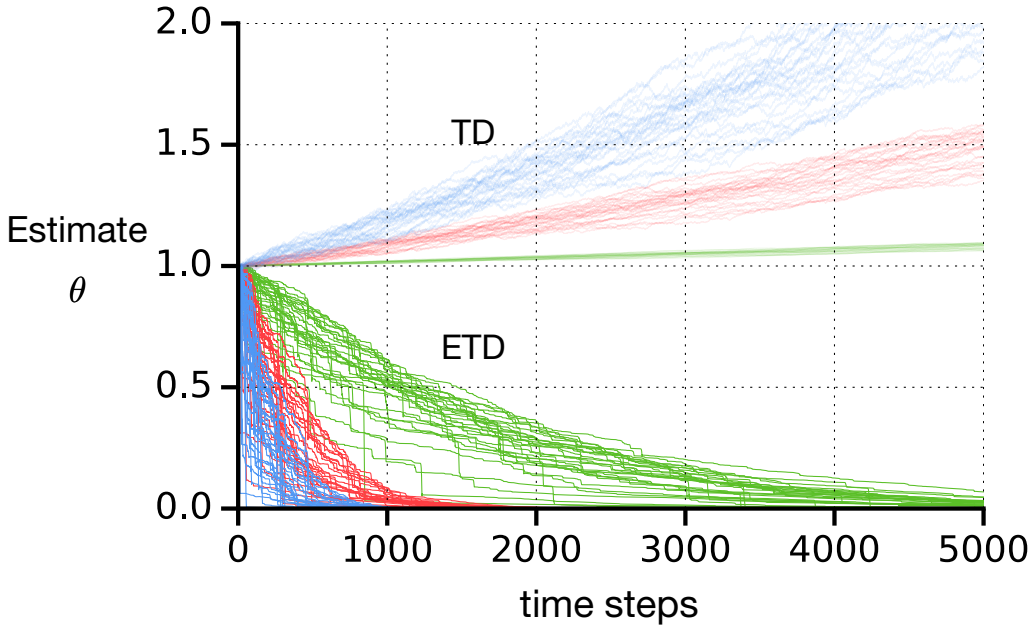

Figure 10.1: Demonstration of stability of ETD(λ) on an off-policy prediction task where TD(λ) is unstable. The three different groups of curves with different colors for each method are for three different step sizes: 0.001 (blue), 0.005 (red), and 0.01 (green).

- *feature function* $\boldsymbol{\phi} : \mathcal{S} \to \mathbb{R}^n$ *s.t. the matrix* $\boldsymbol{\Phi} \in \mathbb{R}^{|\mathcal{S}| \times n}$ *with the* $\boldsymbol{\phi}(s)$ *as its rows has linearly independent columns,*

the $\mathbf{A}$ *matrix of emphatic TD*(λ) *(as given by (10.22–10.26), and assuming the existence of* $\mathrm{E}_0\left[F_t | S_t = s\right]$ *and* $\mathrm{E}_0\left[\mathbf{e}_t | S_t = s\right]$, $\forall s \in \mathcal{S}$*),*

$$\mathbf{A} = \mathrm{E}_0\left[\mathbf{A}_t\right] = \mathrm{E}_0\left[\mathbf{e}_t \left(\boldsymbol{\phi}_t - \gamma_{t+1}\boldsymbol{\phi}_{t+1}\right)^\top\right] = \boldsymbol{\Phi}^\top \mathbf{M}(\mathbf{I} - \mathbf{P}_\pi^\lambda)\boldsymbol{\Phi}, \tag{10.27}$$

is positive definite. Thus ETD(λ) *is stable.*

10.3 Experimental Results

In this section, we experimentally show that ETD is stable on the tasks where TD(λ) is shown to be unstable in the previous chapter.

There are three tasks in total for three different scenarios, off-policy updating, state-dependent bootstrapping, and selective updating, where TD(λ) was shown to be unstable. Figure 10.1, 10.2, and 10.3 show the results on these tasks, respectively.

We also run ETD(λ) on the six prediction tasks from the previous chapter, where TD(λ) was shown to oscillate. The results are shown in Figure 10.4. Although TD(λ) oscillated widely in these tasks, the oscillation was largely diminished with ETD. However, ETD was not completely oscillation free and exhibited a small dip in the curves of some of the tasks.

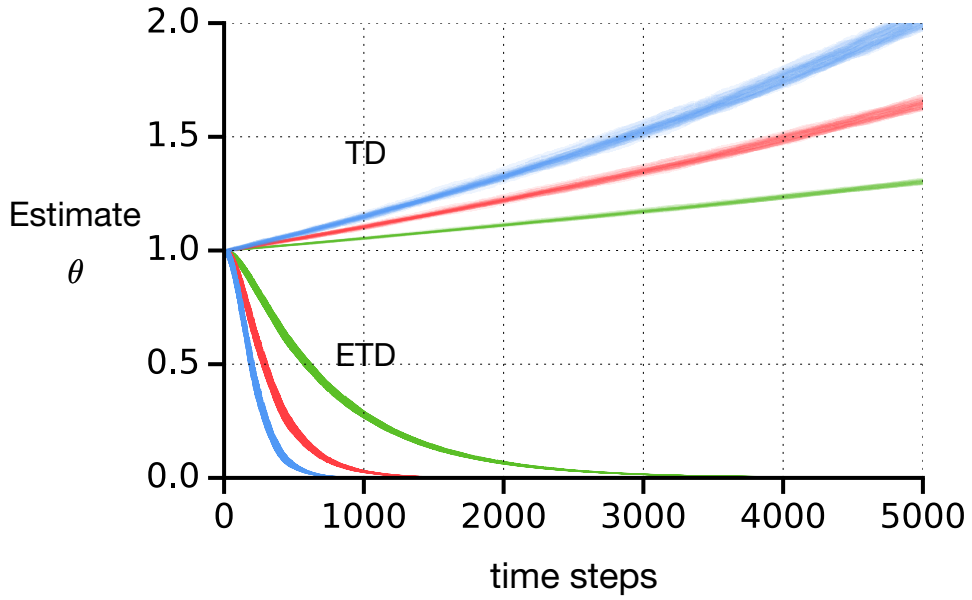

Figure 10.2: Demonstration of stability of ETD(λ) on an on-policy prediction task with state-dependent bootstrapping where TD(λ) is unstable. The three different groups of curves with different colors for each method are for three different step sizes: 0.001 (blue), 0.002 (red), and 0.003 (green).

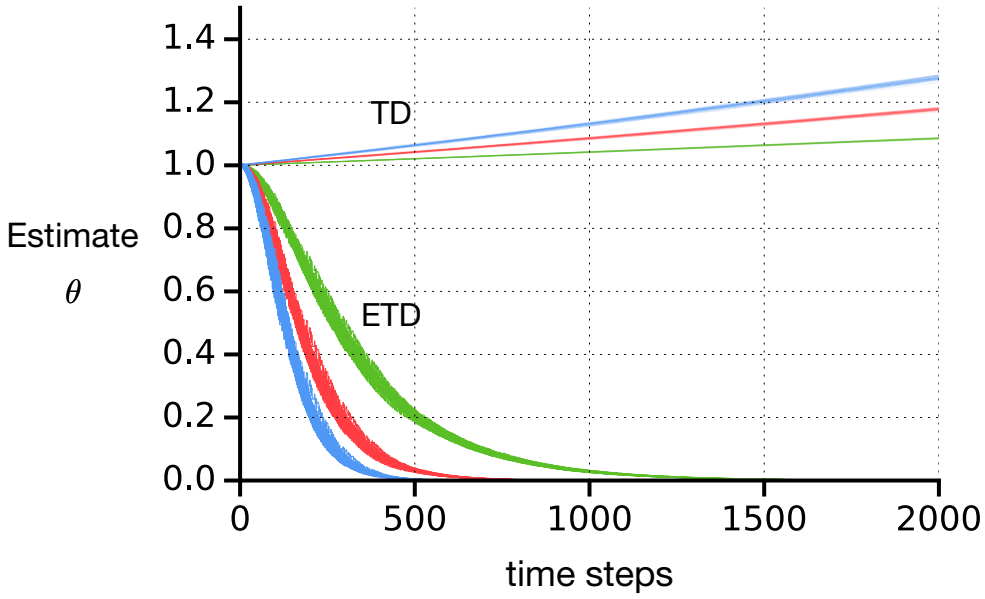

Figure 10.3: Demonstration of stability of ETD(λ) on an on-policy prediction task with selective updating where TD(λ) is unstable. The three different groups of curves with different colors for each method are for three different step sizes: 0.001 (blue), 0.002 (red), and 0.003 (green).

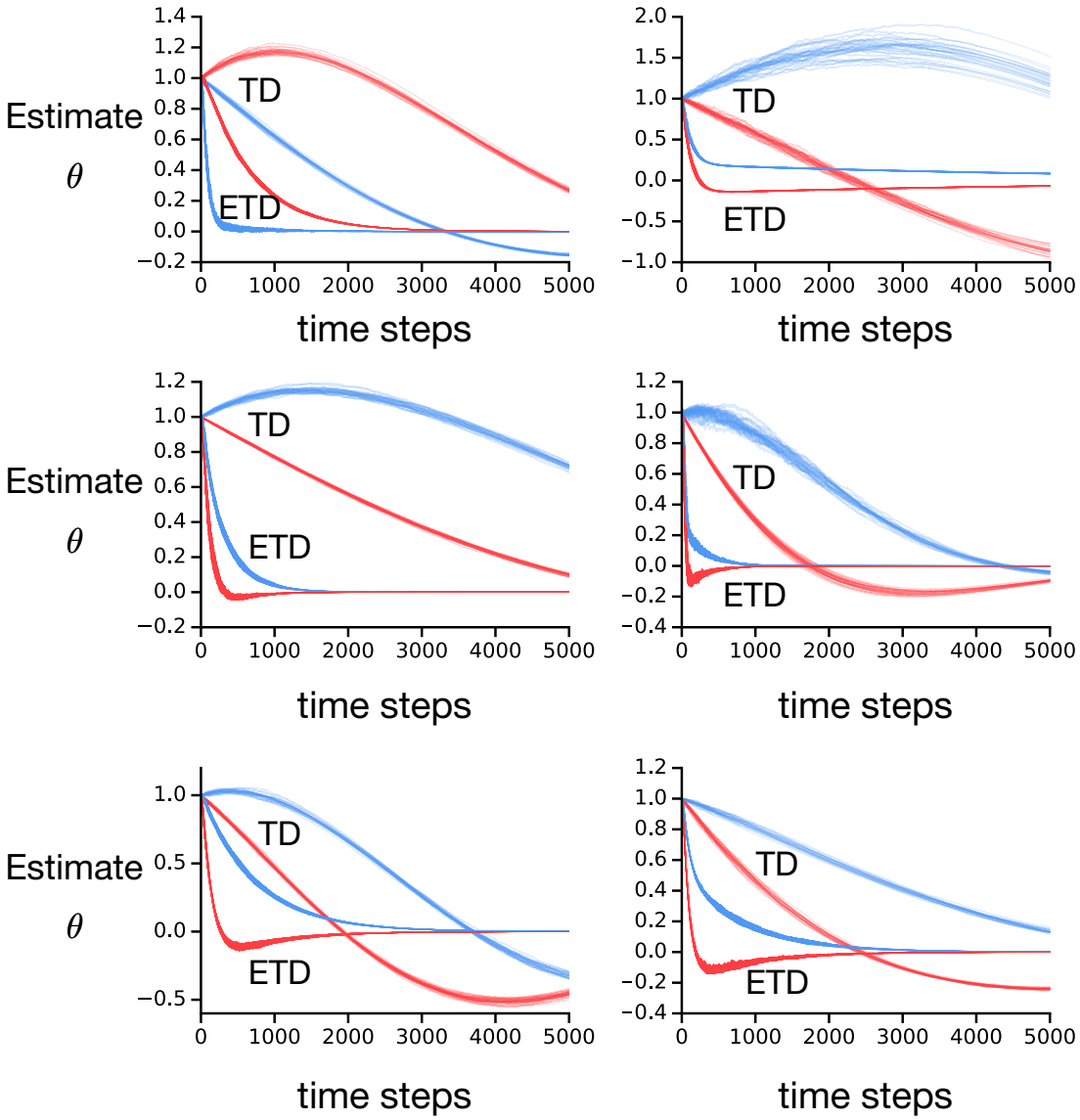

Figure 10.4: ETD oscillates less compared to TD on six prediction tasks. The top two curves of the left-side of each plot belong to TD, and the other two belong to ETD. Although TD oscillated widely, ETD largely diminished the oscillation by emphatic updates. The two curves of each algorithm in each plot are the two elements of the parameter vector $\boldsymbol{\theta}$.

10.4 Conclusions

In this chapter, we developed the first stable off-policy TD algorithm that requires only a single step-size parameter. To achieve this, we utilized our understanding of the issue of instability in terms of the key matrix in the expected update of TD algorithms, which we developed in the previous chapter. Although there can be different ways of achieving stability, we took a simplistic approach where all we require is to ensure that the key matrix is positive definite. We provided a principled way of ensuring a positive definite key matrix, which can be achieved by warping the update distribution, resulting in the ETD(λ) algorithm. We showed empirically that ETD is stable in the same prediction tasks where TD was shown to be unstable in the previous chapter. Finally, we also showed that ETD oscillates less compared to TD in six different prediction tasks.

Chapter 11

Representation Search Through Generate and Test[1]

If off-policy predictions can be learned in a computationally cheap manner, it would be useful to use them numerously as the units of knowledge representation. However, the number of possible useful predictions that can be made in large environments is far greater than the amount of resources available to us. Therefore, it would be prudent to ask only those predictive questions relevant to the overall goal of the agent.

In this chapter, we simplify the problem of selecting and retaining the most useful predictive questions into an online supervised representation search problem. In this simplified problem, the main task is to solve a supervised learning problem. The learner takes the form of a non-linear network. It maps the observations first to a representation layer with non-linear features and then maps the features linearly to produce the output of the system. Both maps are parameterized, where the parameters of the first map are called the input weights and the parameters of the second map are called the output weights. The output weights are learned using a linear learner, for example, linear stochastic gradient descent or the Least Mean Squares (LMS) algorithm.

The representation search problem here is about choosing the right set of features by changing the input weights. There are many representation-learning algorithms that make changes to the input weights to improve representation learning. Here, our main focus is on the problem of searching over a discrete set of features, as if the search were being performed over a discrete set of predictions. This is only a sub-problem of the overall representation discovery problem because the features are parameterized with real-valued weights and are not bound to remain in a discrete set. However, we aim at this specialized problem because it is more pertinent to selecting among a list of potentially interesting predictions. Moreover, a solution method for this problem may also compliment other representation discovery methods. A crucial property we seek from a solution method for the representation search problem is being strictly incremental. We consider the strongest case for this where the

[1]Some of the core concepts in this chapter are developed in a published paper coauthored by this author (Mahmood & Sutton 2013).

system always contains a constant number of features.

This chapter introduces a solution method for the online representation search problem that generates and selects features in a continual and automatic manner. This method constitutes the final key contribution of this thesis. Our solution method is intuitive and also computationally cheap at the same time. The method continually generates new features and tests whether the existing features are useful. As the total number of features remains constant, the method accommodates new features by replacing the least useful ones. Through this continual process of generation, testing, and retention, our methods seek to find a good set of features. Likewise, we call this method *generate and test*. This chapter serves the purpose of describing the representation search problem in a simplified supervised learning setting, and introducing the generate and test method.

11.1 A Simple Representation Search Problem

In this section, we introduce an online representation search problem for a supervised learning task. This task is adopted from Sutton and Whitehead's (1993) work.

In this task, data arrives as a series of examples. The kth example is presented as a vector of $m = 20$ binary inputs $\boldsymbol{\phi}_k \in [0,1]^{20}$ and a single target output $Y_k \in \mathbb{R}$. The target output is computed by linearly mapping the target feature vector $\mathbf{f}^*_k \in \mathbb{R}^{20}$, which are generated using linear threshold units:

$$[\mathbf{f}^*_k]_i = \begin{cases} 1 & \mathbf{v}^{*\top}_i \boldsymbol{\phi}_k > \tau_i \\ 0 & \text{otherwise} \end{cases}, \tag{11.1}$$

where $\mathbf{v}^*_i$ is the target input weight for the ith feature, and τ_i is the threshold for the ith feature. The input weights are initialized with either $+1$ or -1 randomly. The threshold τ_i is set in such a way that the ith feature activates only when at least β proportion of the input bits matches the prototype of the feature. This can be achieved by setting the thresholds as $\tau_i = m\beta - S_i$, where S_i is the number of negative input weights (-1) for the ith feature. The threshold parameter β of these LTUs was set to 0.6.

The target output Y_k was then generated as a linear map from the target features as $Y_k = \sum_{i=1}^{n} \boldsymbol{\theta}^{*\top} \mathbf{f}^*_k + \epsilon_k$, where $\epsilon_k \sim N(0,1)$ is a random noise, and the elements of the target output weight vector $\boldsymbol{\theta}^*$ was chosen randomly from a normal distribution with zero mean and unit variance. Their values were chosen once and kept fixed for all examples. Figure 11.1 shows the architecture of this base learning system.

Here the task of the base system is to approximate the target output as a function of the inputs in an online manner. The approximator only observes the inputs and the outputs. If the approximator uses exactly the same network as the target network, then the MSE performance of the learner would be at minimum, which is 1 in this setting.

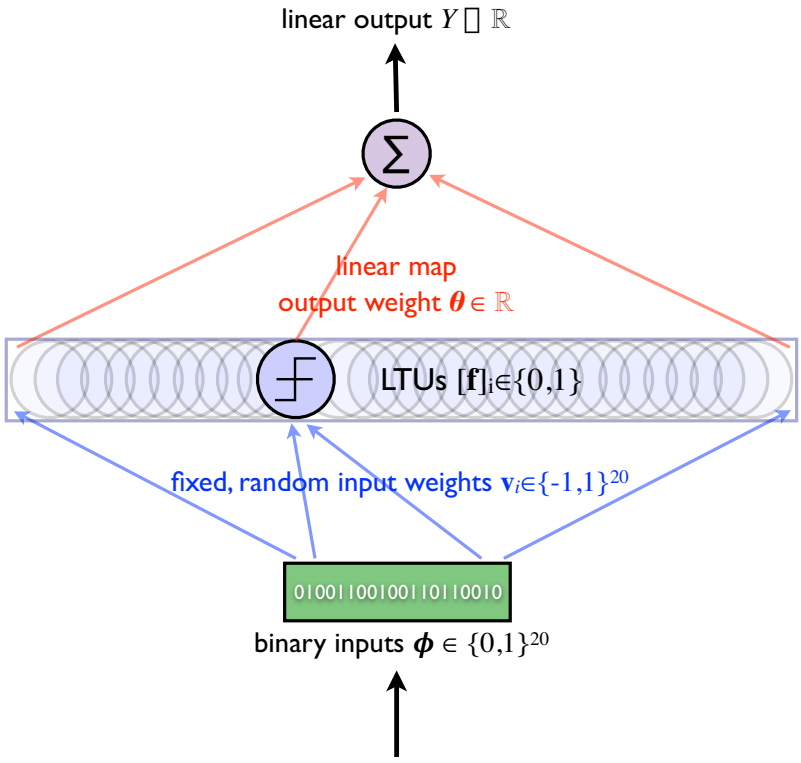

Figure 11.1: The general architecture of the base learning system for the online representation search problem. A binary input vector is nonlinearly mapped into an expanded feature representation. The features are linear threshold units, which are linearly mapped to produce a scalar output. The base system learns the output weights whereas representation search learns the input weights.

11.2 Search Through a Large Number of Random Features

Here, we describe a simple solution method for the representation search problem. The approximator uses a similar network as the target network, that is, it consists of a non-linear input map through LTU features $\mathbf{f}_k$ and an output map from the features to the output. As the approximator has no way of knowing exactly how many and which LTU features are being used in the target network, it contains a large number of them, which are generated randomly. Having a large number of features is the core idea utilized by this solution method. The motivation behind this idea is that the more features the approximator network contains, the more likely it is that some of them will be similar to the target features, and in that sense, it performs a form of search.

The base system approximates the target output as a non-linear function of the inputs. An additional "bias feature" is added to the representation, which always has a value of 1. The input weights $\mathbf{v}_k$ are initialized randomly from $\{-1, +1\}$ and kept fixed. Therefore, the features in the approximator neither increase in number nor alter in terms of its mapping,

limiting the search of this approximator only to the time of initialization. We refer to the representation of this approximator as *the fixed representation.*

The output weights $\boldsymbol{\theta}_k$ are initialized to zero, and updated for each example using the Least Mean Squares (LMS) algorithm:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha\delta_k\mathbf{f}_k, \tag{11.2}$$

Here, δ_k is the estimation error $Y_k - \boldsymbol{\theta}_k^\top \mathbf{f}_k$, and α is the step-size parameter.

The cost for mapping each input vector to a feature vector is $O(mn)$, and producing the linear map from a feature vector to an output costs $O(n)$. Therefore, the total cost of the overall map is $O(mn)$ for each example, that is, proportional to both the number of inputs and features, and remains constant over examples. The computational cost for learning the output weights using LMS is $O(n)$ for each example. Therefore, the total per-example computation used by this approximator is $O(mn)$.

Our objective here is to test the intuition that having more features can make a better approximator. To test this, we test different approximators with different number of features $n \in \left\{100, 200, 10^3, 10^4, 10^5, 10^6\right\}$ and measure the estimated MSE. Each approximator observes the same stream of samples. To estimate their performance, a running average of squared error δ^2 were further averaged over 50 indepedent runs.

Figure 11.2 shows the result of this experiment. It shows that performance increases as the fixed representation contains more features. However, increase in performance decreases as the number of features is increased. Similar results were also found by Sutton and Whitehead (1993) in their work.

11.3 Search Through Generate and Test

In this section, we introduce three representation search methods that search features on an example-by-example basis. Each method searches for features through generate and test. All of the methods use the same generator that generates features randomly. The three methods differ by their testers.

We first describe what is common between these methods. All the methods start with the same representation. After each example is observed, the base system executes its operations once. First the input example is mapped to produce the output, and the output weights are then updated using the LMS algorithm (Eq. 1). When representation search is not used, only these steps are repeated for each example. A representation search method does the following in addition to the operations of the base system. The tester first estimates the utility of each feature. The search method then replaces a small fraction ρ of the features that are least useful with newly generated features. The replacement parameter ρ is a constant and has to be tuned. Input weights v_{ij} of the new features are set with either $+1$ or -1 at random. The output weights w_i of these new features are set to zero. This process is repeated for each example. Note that selecting ρn features does not require sorting all

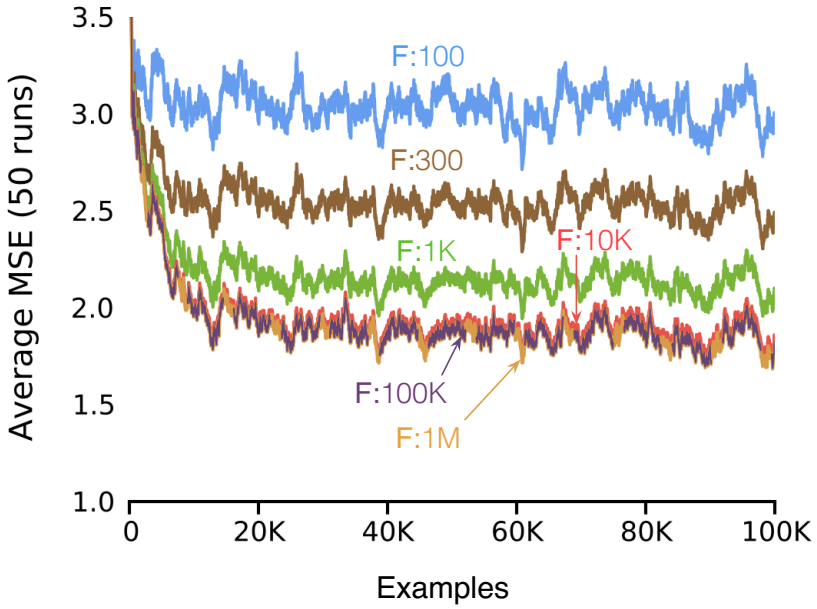

Figure 11.2: The base system with fixed representation performs better in online learning with larger representations. Best performance is achieved by a fixed representation with one million features (F:1M), but the performance increase is negligible compared to the ten times smaller representation (F:100K).

features. It only requires finding the ρnth order statistic and all the order statistics that are smaller, which can be computed in $O(n)$ (Blum et al. 1973). Generating ρn features randomly requires $O(\rho nm)$ computation. Note that ρ is a small fraction.

Our three methods have three different testers. Our first tester uses the magnitude of the instantaneous output weight as an estimate of the utility of each feature. This is not an unreasonable choice, because the magnitude of the output weights is, to some extent, representative of how much each feature contributes to the approximation of the output. When magnitudes of the features are of the same scale, then the higher the output-weight magnitude is, the more useful the feature is likely to be. Features that are newly generated will have zero output weights, and will most likely become eligible for replacement on the next example, which will be undesirable. In order to prevent this, we calculate the age a_i of each feature, which stands for how many examples are observed since the feature is generated. A feature is not replaced as long as its age is less than a maturity threshold μ. Therefore, the selection of ρn least-useful features occurs only among the features for which $a_i \geq \mu$. The maturity threshold μ is a tunable parameter. The age statistics a_i can be kept and updated using $O(n)$ time and memory complexity. Table 11.1 describes all the steps for the first method.

Our second tester uses the trace of the past weight magnitudes instead of the instanta-

Algorithm Simple Online Representation Search

Initialization:
Set input weights $\mathbf{v}_i, i = 1...n$ randomly
Set output weights $\boldsymbol{\theta}$ to zero
*Set age a_i of each feature to zero
*Set replacement rate ρ, e.g., to $\frac{1}{200}$ per example
*Set maturity threshold μ as desired
for each example $(\boldsymbol{\phi}, Y)$ **do**
  Map inputs to features: $[\mathbf{f}]_i \leftarrow \text{LTU}(\mathbf{v}_i^\top \boldsymbol{\phi})$, for $i = 1 \ldots, n$
  Map features to output linearly: $\boldsymbol{\theta}^\top \mathbf{f}$; $[\mathbf{f}]_0 = 1$
  Update each $\boldsymbol{\theta}$ using LMS, Eq. (11.2)
  *Increase each a_i by one
  *Find $n\rho$ features with smallest $|[\boldsymbol{\theta}]_i|$ s.t. $a_i \geq \mu$
  *Set $\mathbf{v}_i$ randomly for those $n\rho$ features
  *Set $[\boldsymbol{\theta}]_i \leftarrow 0$ for those $n\rho$ features
  *Set $a_i \leftarrow 0$ for those $n\rho$ features
end for
Steps marked by asterisks (*) constitute search. When these steps are not executed, this algorithm becomes the base system with fixed representation.

Table 11.1: A simple online representation search method through generate and test.

neous ones. The trace is estimated as an exponential moving average, which can be updated incrementally. Instead of using an age statistic for each feature, the trace of a newly generated feature is initialized using a particular order statistic of all the existing traces (e.g., the median of all traces), so that newly generated features do not get replaced immediately. If a feature is irrelevant, its weight will have a near-zero value, and its trace will also get smaller with time, making the feature eligible for replacement. The decay rate of the exponential average and the order statistic for initializing the traces are tunable.

Our third tester uses the instantaneous output weight magnitudes for estimating the utility, but also uses learned step sizes as measures of how reliable the weight estimates are. No age statistic is used in this tester. We use the Autostep method by Mahmood et al. (2012) that learns one step size for each feature online without requiring any tuning of its parameters. Higher confidence is ascribed to a weight estimate if the corresponding feature has a smaller step size. The initial step size of a newly generated feature is set to a particular order statistic of all step sizes. A feature is eligible for replacement only if its step size is smaller than that statistic. The order statistic is a tunable parameter.

Per-example computation cost for all three testers is $O(n)$, hence our online representation search methods use a total of $O(n) + O(\rho nm)$ computation. Therefore, the order of per-example computation of the representation search methods is not more than that of the base system. If we choose ρ always to be less than $1/m$, then the total cost becomes $O(n)$.

Moreover, each tester overcomes the difficulty of reliably estimating the feature utility by using different measures (age statistics, traces and step sizes).

we empirically investigate whether our representation search methods are effective in improving representations. The base system performs a supervised regression task, and the task of a representation search method is to improve the performance by searching and accumulating better features.

Data in our experiment was generated through simulation as a series examples of 20-dimensional i.i.d. input vectors (i.e., $m = 20$) and a scalar target output. Inputs were binary, chosen randomly between zero and one with equal probability. The target output was computed by linearly combining 20 target features, which were generated from the inputs using 20 fixed random LTUs. The threshold parameter β of these LTUs was set to 0.6. The target output y_k was then generated as a linear map from the target features $\mathbf{f}_k^*$ as $y_k = \boldsymbol{\theta}^{*\top}\mathbf{f}_k^* + \epsilon_k$, where $\epsilon_k \sim N(0,1)$ is a random noise. The target output weights $\boldsymbol{\theta}^*$ were randomly chosen from a normal distribution with zero mean and unit variance. Their values were chosen once and kept fixed for all examples. The learner only observed the inputs and the outputs. If the features and output weights of the learner are equal to the target features $\mathbf{f}_k^*$ and target output weights $\boldsymbol{\theta}^*$, respectively, then the MSE performance $\mathrm{E}\left[\left(Y_k - \boldsymbol{\theta}^\top \mathbf{f}_k\right)^2\right]$ of the learner would be at minimum, which is 1 in this setting.

For all the methods except the third representation search method, the step-size parameter has been set to $\frac{\gamma}{\lambda_k}$ for the kth example, where $0 < \gamma < 1$ is a small constant, that we refer to as the *effective step-size parameter*. Here, λ_k is an incremental estimate of the expected squared norm of the feature vector $\frac{1}{T}\sum_{k=1}^{T}\mathbf{f}_k^\top\mathbf{f}_k$. The effective step-size parameter γ is set to 0.1 for all the experiments. The replacement rate ρ is set to 1/200, which stands for replacing one feature in every 200 for every example. The rest of the parameters of the representation search methods are roughly tuned.

First we study how well the base system with fixed representations performs with different size of representations. Figure 2 shows the performance of fixed representations with different sizes (from 100 up to one million features) over one hundred thousand examples. Performance is measured as a running estimate of Mean Squared Error (MSE). Performance is averaged over 50 runs. Results show that fixed representations with more features perform better. However, as number of features is increased, the increase in performance becomes smaller and smaller. Similar results were also found by Sutton and Whitehead (1993) in their work on online learning with random representations.

The result of our first representation search method is shown in Figure 11.3 over one million examples. This result is on the task as in the previous section. Performance is measured as an estimate of MSE averaged over last 10,000 examples and 50 runs. The search method performed substantially better than fixed representations and continued to improve as more examples are seen. Performance of the fixed representation with 100

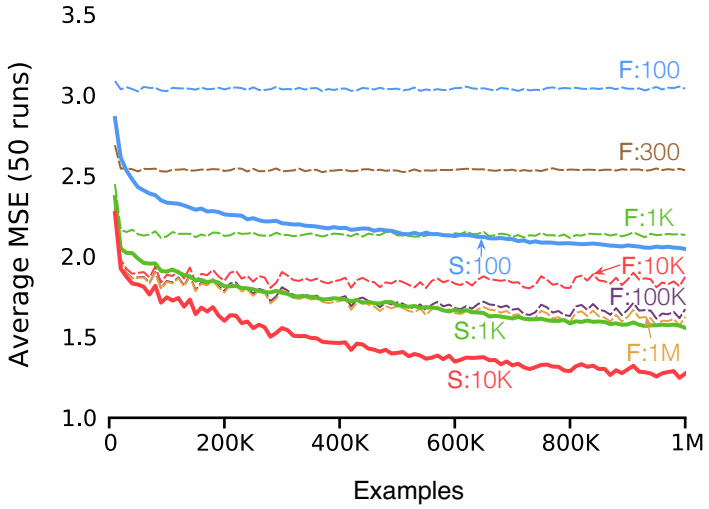

Figure 11.3: A simple representation search method outperforms much larger, fixed representations. This method with 1,000 features (S:1K) outperforms a fixed representation with one million features (F:1M) and continues to improve. Search with larger representations perform even better, approaching the minimum possible value of 1.0.

features (F:100) settled at a certain level, but representation search with the same number of features (S:100) outperformed it at an early stage and continued to improve until the end of the sequence. Representation search with 1,000 features (S:1K) outperformed fixed representation with 1,000 times more features (F:1M).

Figure 11.4 compares the three representation search methods on the same problem as previous. Performance after observing one million examples is plotted against different number of features. The simple tester is outperformed by the other testers. The tester with learned step sizes performed the best.

Figure 11.5 shows the distribution of output-weight magnitudes for a fixed representation and representation search methods with different testers. Two thousand features were used for each representation. These results show that most weights in both fixed representations and search are small. The tester using step sizes had the largest weights of all methods among the top 0.5% of its weights and the smallest weights of all methods among the bottom 20% of its weights.

In the following we pursue a detailed discussion on how a generate and test process for representation search can be developed. A generate and test process has two components: a generator and a tester. A generator generates new features in various ways, for example, by setting the input weights randomly. A tester estimates the utility of each features individually based on the actual performance. A generate and test process uses both a generator and a tester to search the feature space and accumulate the most useful features.

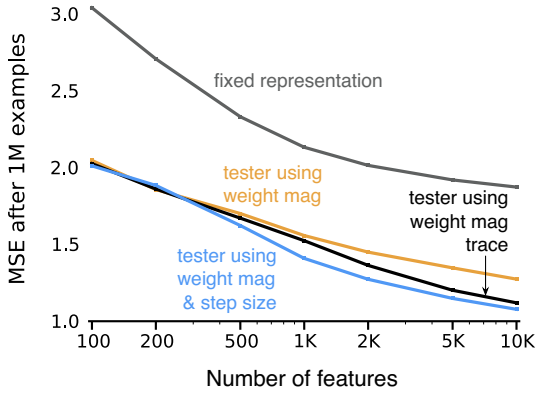

Figure 11.4: The choice of a tester has a significant effect on the performance of representation search. Our simplest tester using weight magnitudes is outperformed by testers that use more reliable estimates of feature utility. However, all of them perform better with more features.

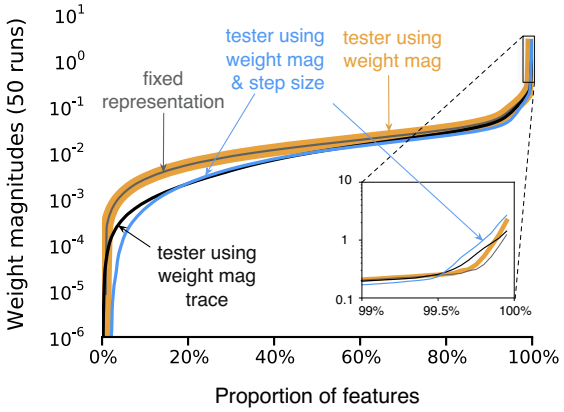

Figure 11.5: The distribution of weight magnitudes (after 1M examples) for a fixed representation and search methods with three different testers. In all cases, most features have small weights, but there are differences at the extremes of very large and very small weights.

In the following, we discuss the issues that can come up in developing a tester and a generator, respectively.

11.4 Discussing Different Combinations of Generate and Test

Here we briefly discuss how a generator and a tester can be combined in different ways when using the generate and test process online. In previous, we adopted a simple and straightforward way of combining these two: select a small fraction from the least useful features identified by the tester and replace them with new features generated by the generator. We found that this process is able to change the representation on every example using a small extra computation.

A different method for combining a generator and a tester can diverge from our original methods in different ways. For example, instead of always choosing the least useful features for replacement, features can also be replaced randomly where their replacement probabilities are based on their utility. Moreover, instead of using the generate and test process on every example, it is also possible to change the representation after several examples are observed. It is also possible to change the representation only after a particular event occurs, for example, if the error does not decrease significantly for a certain number of examples, as in the cascade-correlation method (Fahlman & Lebiere 1990), or if a sudden change in the error level occurs, as in Q-learning with "hidden-unit restarts" (Anderson 1993).

Changing the representation on an example-by-example basis is not only an extreme form of online learning, it is perhaps also a practical way of fulfilling the computational constraints faced by online learning problems with frequent and unending stream of data. When data is arriving at a fairly constant rate (e.g., through robot sensors), the overall system can use a small constant amount of time between two examples to complete all the mappings and learnings that need to be done for the current example. The computation for representation change has to be allotted within this small period of time. Now, if the representation changes only once in a while, the allotted time would be wasted for those times when the representation does not change. It would be desirable, in that case, to amortize the computation for representation change throughout all examples. Our previous results shows that it is possible to change representation on every example and improve the representation effectively at the same time. However, our methods do not differentiate between different examples, and hence are not amortizing an event-based representation change. It is not clear whether there can be a substantial advantage of using infrequent or event-based representation changes over changing the representation on every example in terms of efficiency in performance. If the former is more likely to be efficient in the problems we are interested in, then it would be desirable to amortize them over every example, too. For example, instead of making decisions on whether or not a representation change should occur based on a binary event, the event can be quantified in terms of a per-example intensity, and the representation change can then be proportional to the intensity of the event. In that case, the representation change would be able to use roughly the same amount of computation on every example.

The generate and test process involves several parameters of its own that need to be tuned. For example, all our representation search methods, introduced previously, have a replacement-rate parameter that determines the fraction of all features that would be replaced on every example. There are other parameters too, for example, the maturity threshold, the decay rate for traces and the initial value of the traces. In general, it might be useful to automatically tune some of these parameters. For example, the replacement rate might be automatically tuned to control how many features to replace on each example. If it is possible to detect that further search would not improve the representation or even deteriorate the current representation, the replacement rate can be reduced.

11.5 Conclusions

In this chapter, we introduced an online representation search problem as simplification of the problem of searching for useful predictions. This opens up the possibility of curating predictions as the features of knowledge representation. However, we do not explore this possibility in this thesis but rather focus on this simplified setup. In this simple representation search problem, the learner solved a supervised learning task while continually searching over a discrete set of features in a strictly incremental manner. We provided some

solution methods based on a simple idea we call generate and test, where features were continually generated and tested for their usefulness. These search methods constitute a powerful approach to representation discovery complementing other representation learning algorithms in a continual learning setting.

Chapter 12

Search as a Complementary Approach to Representation Learning[1]

In this chapter, we explore how our online representation search methods can be combined with some of the existing representation learning methods and how the combination may lead to performance improvement. We chose one of the online search methods we developed in the previous chapter for this purpose. We explore how it interacts with an unsupervised representation learning method by Sutton and Whitehead (1993) and a supervised representation learning method known as the backpropagation method (Rumelhart et al. 1986).

12.1 Search with Unsupervised Learning

In this section, we discuss the interplay between representation search and a widely popular representation learning strategy known as *unsupervised feature learning*. We discuss how representation search through generate and test can be combined with unsupervised feature learning methods in an online manner. We point out two benefits of doing that in a continual learning problem. First, as in supervised gradient-descent learning, the continual injection of feature variations through generate and test can also increase the generalization ability of unsupervised learning. Second, a tester can be used to provide supervised feedback to unsupervised feature learning and thus improve performance. We show some preliminary results supporting them, and propose further investigations to fully explore them.

Unsupervised Feature Learning

First, we briefly describe the core ideas of unsupervised feature learning. Unsupervised feature learning stands for a broad class of representation learning methods where the representation of input data is learned without using any feedback of the performance on the

[1]Some of the core concepts in this chapter are developed in a published paper coauthored by this author (Mahmood & Sutton 2013).

original learning task (Bengio et al. 2012). For example, in a classification task, the performance depends on how well the learner predicts the correct label of the input data. An unsupervised learning method disregards the labels, and uses information from the input data only. Unsupervised criteria such as sparsity (Olshausen & Field 1997, Lee et al. 2007), statistical independence (Comon 1994) and reproduction of the input data (Hinton & Salakhutdinov 2006, Bengio et.al. 2007a, LeCun & Bengio 2007) are commonly used for learning the features. For example, in autoencoders (Bengio et al. 2007a, Hinton & Salakhutdinov 2006) features are learned in a way so that the input data can be reconstructed from the features. An input vector is mapped into a feature representation, and the target outputs are also exactly the same as the elements of the input vector. Both input and output weights are learned through error backpropagation rule so that the autoencoder can approximate the inputs. After the unsupervised learning is over, the output weights are discarded, and the learned features are then used to solve the original classification task. The map from the features to the correct labels are learned at this stage. Supervised gradient-descent learning through error backpropagation can also be used to fine tune the features. A related concept is called a restricted Boltzmann machine (RBM) (Smolensky 1986, Hinton & Salakhutdinov 2006) where a probability distribution of the input data is learned.

Unsupervised feature learning have been successful in learning networks with many layers, a sub-field of machine learning often known as *deep learning* (Bengio et al. 2012, Erhan et al. 2010). Deep networks can often represent concepts more compactly and efficiently than shallow networks (Bengio & LeCun 2007b). They can use several levels of abstractions to capture features that are invariant to low level transformations of the input data. However, learning features of deep networks is difficult. Supervised gradient-descent learning through error backpropagation does not perform well in deep networks. Many bad local minima can exist in deep networks, and gradient-descent learning can easily become susceptible to them. Moreover, gradients of the early layers computed through backpropagation become very small in deep networks, a problem known as the *vanishing gradient problem* (Bengio et al. 1994). On the other hand, unsupervised feature learning have been successful in learning the features of deep networks. One of the most common strategies is to develop a deep network by stacking multiple autoencoders. Each hidden layer is learned through a standalone autoencoder, and the learned features are then fed to the autoencoder of the next hidden layer as inputs, and so on.

Unsupervised feature learning methods are almost always used in a batch based on a fixed set of data. They are often viewed as a tool for finding a good set of initial weights for supervised gradient-descent learning, and hence the process is often also called *unsupervised pre-training*. Although the initial weights are extremely important in a short-lived batch learning task, their importance diminishes with time in a continual learning task. When learning occurs unendingly in an online learning setting, unsupervised learning methods

have to be used continually and on an example-by-example basis. Zhou et al. (2012) exemplified how unsupervised learning methods can be interleaved with supervised gradient-descent learning in order to use them continually. However, their method operates on mini batches, not on an example-by-example basis. A detailed study is needed on how unsupervised learning can be conducted fully online.

Continual Injection of Variations in Unsupervised Feature Learning

When unsupervised learning is conducted online, continual injection of variations to features may have a similar effect as initial weights have in batch learning. Unsupervised learning typically starts with random initialization of weights, which provides an initial search of features. Online representation search through generate and test in a way amortizes this search by injecting random variations to the representation continually. When unsupervised learning is run online, this continued search may become useful. This can in fact help unsupervised learning to avoid local minima. This constant source of new features might also be helpful to increase the generalization ability of unsupervised learning.

The role of continual injection of variations can be essential in non-stationary or sequential learning problems. When the learner is faced with a new task, there is no straightforward way to detect the change so that the weights can be reinitialized. Even if it is possible, it would forget all the learnings from the previous task, and hence would be undesirable. The continual injection of random variations that representation search introduces to a small fraction of the features can be a reasonable remedy to this problem.

A potential problem may arise in unsupervised feature learning methods due to the fact that features learned by them are not tested. These methods learn features based on a criteria that is not directly related to the original learning task. Although, features learned by unsupervised learning have been found to be useful, however there is no guarantee that features learned in this manner would remain useful in general. As unsupervised learning disregards the feedback from the actual performance, it may produce many irrelevant features together with the useful ones. When unsupervised learning is used online for learning continually, the continual process of generating features through unsupervised learning may also destroy some existing useful features in the process. Whenever, a less useful or irrelevant feature would fulfil the unsupervised criteria more than a useful feature, the unsupervised learning method would prefer the irrelevant feature over the useful one. It has no way to tell which one is more relevant to the original learning task. In continual learning tasks, it is not desirable to disregard the usefulness of features for too long.

As a tester in our generate and test process sifts through the feature set and determines the usefulness of each feature, an unsupervised learning method might use it to get supervised feedbacks. For example, a tester might be used to identify a fraction of the most useful features on a given example and protect them for that step. Unsupervised learning can then be applied only to the rest of the features. It may enable unsupervised learning to

learn features selectively and accumulate the useful ones over times. This can also be viewed as a form of representation search through generate and test where features are generated through unsupervised learning, and they replace the least useful features identified by a tester. This way, this combination may form a symbiosis between these two distinct ideas of representation learning.

Empirical Study

Here we conduct a preliminary study to investigate the interplay between representation search and unsupervised learning in an online learning task. Specifically, we run experiments to verify whether the two proposed interactions of representation search in unsupervised learning we discussed above, continual injection of variations and supervised feedback through a tester, are indeed useful.

As part of the base system, we use an unsupervised learning method that learns sparse features in an online manner, while the original learning task is also being solved at the same time. The original learning task is an online supervised regression problem. The base system, on which representation search would be applied, would take an input example, map it to a feature representation of LTUs, and then would map linearly to an output. The output weights are updated using the LMS rule (Eq. 11.2). The base system then uses the online unsupervised learning method to adjust the sparsity of the LTUs on that example. This process would repeat for each example. Its per-example complexity has to scale linearly with the number of total weights.

Our chosen unsupervised learning method, developed by Sutton and Whitehead (1993), applies two techniques, one after another. These techniques adjust the feature activations both across examples and among features. These techniques are fully online, and their per-example complexity scale linearly with the number of total weights.

The first technique modifies the frequency of the activation of each feature across examples. For this, it computes the running estimate of the activation frequency of each feature. The threshold θ_i is incrementally changed so that the estimated frequency is within a small bound around a target frequency. This unsupervised learning technique has two parameters: the target frequency and the tolerance bound.

The second technique modifies the density or the total activation of all features so that it achieves a target density. If the density of the feature set at any moment goes below a certain limit of the target density, one of the inactivate features is chosen with a small probability (0.0001), and then one of its input weights that do not match with their corresponding input bits is selected randomly. The sign of that input weight is then flipped. The corresponding threshold θ_i is then decreased by 1 to reduce the effect of this change on the frequency of that feature. If the density of the feature set at any moment goes above a certain limit of the target density, one of the activate features is chosen with a small probability (again, 0.0001), and then one of the input weights that match with their corresponding input bits

is selected randomly. The sign of that input weight is then flipped. The threshold is also increased by 1 in this case. This technique in effect adjusts the features so that at least some of them responds to the inputs to some degree. This technique has two parameters: the target density and the tolerance bound.

We use two variations of representation search methods on top of the base system. In the first variation, a tester is used to identify a fraction of the most useful features on each example. There features are protected from being altered by unsupervised learning on that example. The unsupervised learning method is applied only on the rest of the features. For example, if a feature is protected, its threshold and input weights are not adjusted to match the target frequency and density. Its usefulness would be preferred over its sparsity. However, the feature itself might have been arrived at by adjusting the sparsity at the beginning. We use the trace of the weight magnitudes as the tester. Here, additional parameters due to search are the trace parameter and the selection rate which denotes the percentage of features that are protected on each example.

The second variation of representation search includes every step from the first variation. Additionally, it replaces a small fraction of the least useful features with randomly generated features. This step is exactly the same as our representation search methods through generate and test described in the previous chapter. Additional parameters it introduces are the replacement rate and the order statistic of all traces with which the trace of a new feature is initialized.

Representation search would be operated with unsupervised learning in the following way. When an example is presented, it is mapped to the output through the feature representation, and the output weights are learned. Then a fraction of the best features is identified. The unsupervised learning method is applied on the rest of the features to update their thresholds and input weights. If the second variation is used, then a small fraction of the least useful features is also replaced with randomly generated features. These steps are repeated for each example. Per-example computation of all these steps is $O(mn)$, where m is the number of inputs and n is the number of features.

We used an online supervised learning problem similar to that in section 3. The target outputs were generated through the weighted sum of 100 target LTUs as functions of the inputs, generated randomly. Threshold parameter β of the target LTUs were set randomly between 0.0 and 1.0. The target output weights were generated from a standard normal distribution. Inputs were binary as previous. In section 3, the activation probability of inputs was 0.5, so all the input vectors were equally likely. Unsupervised learning would not have any significant advantage in this kind of problems. In the current experiment, we set the activation probability of inputs to 0.2. This way, some input vectors were more likely than others, and unsupervised learning would have an advantage over fixed random representations.

For the unsupervised learning method, we experimented with different target frequencies

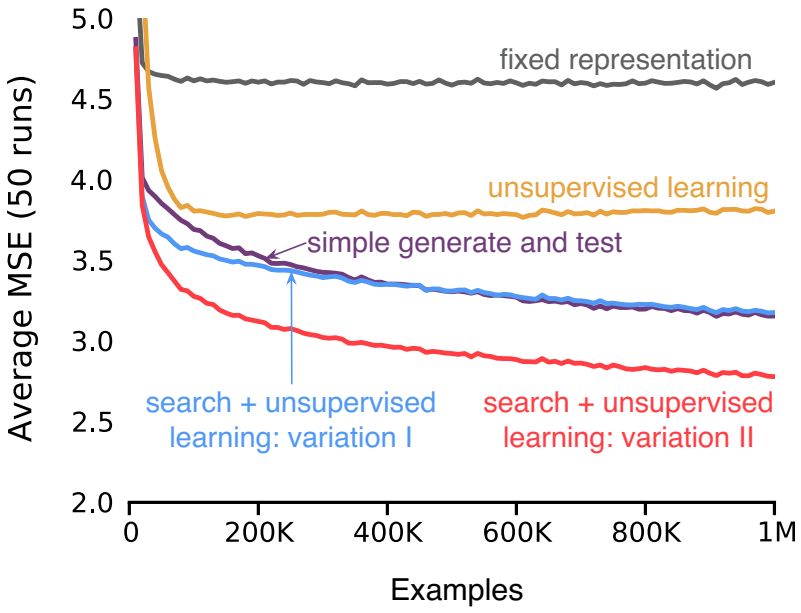

Figure 12.1: Representation search improves the performance of unsupervised learning. When some of the useful features are preserved through representation search, and the unsupervised learning method are applied only on the rest of the features (variation I), it worked better than applying the unsupervised learning method on all features. When the random generation of features was included to the first variation (variation II), it performed the best among all variations.

and target densities, both between 0.0 and 1.0. Following Sutton and Whitehead (1993), The tolerance bound around the target frequency was always set to 0.05. Unlike the experiments in the previous chapter, we used different threshold parameter β for each different features, chosen randomly between 0.0 and 1.0. The influence of the threshold parameter β on the threshold θ remains only during the initial period, and the threshold θ eventually gets modified by frequency adaptation.

For the representation search methods, the selection rate was tuned among $[0.0\,, 0.1, 0.2, \ldots, 1.0]$ and the replacement rate was tuned among $\left[5 \times 10^{-3}\,,\ 2 \times 10^{-3}, 1 \times 10^{-3}, 5 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-4}, 5 \times 10^{-5},\ 2 \times 10^{-5}\right]$. The trace rate was set to 0.001 and the order statistic was set to the median.

From our experiments we find that representation search has the effect on unsupervised learning as we anticipated. The combination of search and unsupervised learning outperformed both of their standalone variation. Moreover, search enabled unsupervised learning to improve the representation continually with more data, whereas when only the unsupervised learning method was used, its advantage was only during the initial period, after which performance settled at a level. Figure 12.1 shows the results. When the first

variation of search is combined with the unsupervised learning method, it outperformed the standalone unsupervised learning method and performed as well as the simple generate and test search that replaced the least useful features with randomly generated features on a base system that were only learning the output weights. When the second variation of search was added to the unsupervised learning method, it increased the performance over the first variation and performed the best. Both in the first and the second variation, the best selection rate was 60%, which signifies that when applying unsupervised learning, a large portion of the features needed protection. Note that, both variations of the combination are improving the representation and performance, continually with more data, whereas the standalone unsupervised learning method was not effective after an initial period of learning. It signifies that, by applying the unsupervised learning method always on the least useful features, more generalization ability could be extracted, and the continual injection of variation boosted this generalization capacity further.

12.2 Search with Supervised Learning

In this section, we discuss the effect of using online representation search on a base system that is already learning its representation using supervised gradient-descent (GD) learning through error backpropagation.

The objective and the computational efficiency of supervised GD learning is similar to those of online representation search. When backpropagation is used as a stochastic gradient descent method, its per-example computation remains constant over time and scales linearly with the number of total weights, which is in the same order as that of the overall input-output map. Moreover, supervised GD method learns the hidden units or features through error backpropagation in order to optimize the supervised performance. Therefore, both GD learning and representation search through generate and test are fully compatible with online learning, and they target at improving the performance of the original learning task.

There are some key differences that distinguish supervised GD learning from representation search. GD learns the features using a directional approach; it greedily updates the weights toward the gradient descent direction. On the other hand, the generate-and-test approach searches and accumulates good features in an undirected and retrospective way. Unlike GD learning, representation search does not commit to any particular way of generating features, but may generate them in many different ways and keeps only those features that has been tested and proved useful on the original learning task. Therefore, these two methods are quite different functionally, and they may have different strengths and shortcomings.

We investigate here how the combination of both interplays with each other, and whether the shortcoming of one can be alleviated by the inclusion of the other. A shortcoming of representation search is that it is a slow process. As it searches and accumulates features retrospectively, it requires a lot of time to try numerous possibilities and identify the most

useful features. On the other hand, GD is comparatively faster, as it conducts directional fine tuning of all the weights simultaneously. For many problems, it leads to a fast improvement to the representation. However, it does not necessarily mean that combining them together would benefit both. It may happen that the fast change of features done by GD learning completely overwhelms the representation search process and makes the effect of search either insignificant or futile. On the other hand, the variations that search injects to the representation may also hamper GD learning.

Two Issues with Supervised GD Learning

We figure out two issues with supervised GD learning which indicate that combining representation search and supervised GD learning together can be useful. The first issues with GD learning is that it is largely sensitive to the initialization of the weights. Some configurations of initial weights, for example setting all weights to zero including the input weights, can render the method to be totally ineffective. To avoid potentially harmful configurations, weights are commonly initialized with different values at random. Random initialization is so common in gradient-descent learning that it is often seen as a part of it. However, it is worthy pointing out that random initialization is also a form of search, where the process of generating random features occur only initially, and gradient descent uses this initial search for its benefit. This initial search attempts to sprinkle the features at widely different places of the feature space with the hope that at least some of them are placed close to favorable areas. GD learning then fine tunes those features to improve the solution. Generate and test can be seen as a continuation of this random variation through continual injection. If the initial search provides enough variations, then continual injection of random variation during GD learning may not provide any extra advantage. However, often the objective function has many poor local optima, and an initial search of features may not cover enough areas of the space. In this case, occasional injection of random variations through generate and test may provide a way to continue exploring the space and a chance to improve the representation. Moreover, if the solution is not stationary and needs tracking, an initial search would hardly suffice, and a continual search in the feature space would be useful for the tracking. Therefore, we recognize that, GD learning already uses a form search, and combining continual search with it is a reasonable idea.

The second issue with GD learning is that it is ineffective in retaining the learning acquired from a previous task while learning on a new task. When different tasks are posed sequentially, its performance on previous tasks degrades abruptly. This problem is known as the *catastrophic interference* problem (McCloskey & Cohen 1989, Ratcliff 1990, French 1999). It is not unique to backpropagation learning, but rather a common characteristic to many learning methods for connectionist networks. Difficulty in retaining the acquired knowledge from an early task is not uncommon in human learning, a phenomenon known as *retroactive interference* (Barnes & Underwood 1959). However, forgetting in artificial

connectionist networks is much more abrupt and catastrophic. McCloskey and Cohen (1989) demonstrated that after only a moderate amount of learning on the second task, the network forgets almost completely about the first task. This is largely due to the distributed nature of representations in connectionist networks. A connection is not tightly bound to a particular kind of patterns, but it is rather shared among various kinds of inputs. Therefore, the weight of a connection learned from one task soon gets recycled by successive tasks. This problem becomes worse when the backpropagation learning rule is used. The update of the inner weights of a feature through error backpropagation is proportional to the output weight of that feature. As the magnitude of the output weight often signifies the importance of the feature, the backpropagation update of inner weights tends to recycle the most useful features the quickest (Sutton 1986). Consequently, during the second learning task, the backpropagation rule quickly changes the features that contributed the most to the first learning task. Long-lived continual learning problems such as life-long learning of robots are often viewed as the sequential learning of many related tasks, for example, learning to walk before learning to run. If GD learning is used as it is for continual learning, the catastrophic interference would make it practically ineffective.

Several solutions have been proposed to amend the problem of catastrophic interference in connectionist networks. One solution is to interleave examples from the two learning tasks instead of presenting them sequentially (Ratchliff 1990, Robins 1995). In that case, the representation will learn features that are relevant to both tasks together, and neither of them will be forgotten. An issue with this solution is that it violates the continual nature of learning and assumes that all learning tasks will be available at the same time. But tasks are often sequential by construct in continual learning problems, and a task may appear only after another one is learned (e.g., learning to walk before learning to run). This issue is often addressed by using two networks at a time (Robins 1995, Robins & Frean 1998, French 1999). One of the networks is used to learn based on the previous task. When the first task is over, this network is used to produce pseudo-patterns based on the first task, while the other network is learned on the new task. The pseudo-patterns are interleaved with the examples from the new task and fed into the second network. These solutions do not prevent the catastrophic interference in the learner, but rather attempt to cure it by changing the learning model or the tasks altogether.

McCloskey and Cohen (1989), in one of the earliest works on catastrophic interference problem, hinted about an obvious and straightforward solution to this problem, which appears to be closely related to representation search. They proposed that the features that are already learned based on previous tasks should be identified, and the learner should avoid using them in subsequent tasks. They immediately acknowledged that it is not known how to distinguish between a learned feature and a feature that is not utilized enough by the learner. This may remind the reader of testers which is a core part of representation search through generate and test. Testers are specifically used to identify

useful features. Moreover, the problem of distinguishing between features that are learned reliably and features that are yet to be learned is one of the main concerns when generate and test is used online and is already taken care of by the testers (see Section 4). Therefore, it might be possible to use a tester to identify the useful features and protect them from being modified abruptly during the subsequent tasks. This solution is more natural than the other solutions as it does not change the model or the task, but rather applies small modifications to the existing learning rule in order to alleviate the problem.

Role of Continual Feature Generation in Supervied GD Learning

From the above discussion, it turns out that the most important distinction between supervised GD learning and representation search is that, despite both of them being computationally suitable for online learning, search through generate and test naturally fits to continual learning, whereas supervised GD learning evidently have issues that may prevent it from being used effectively in continual learning problems. When posed with a sequence of many tasks, supervised GD learning by itself cannot accumulate knowledge or gradually improve the representation. Even if the tasks are related, where many features can be reused, backpropagation update will de-learn them and relearn again, hampering the learning severely.

The role of injecting random variations in GD learning may also become more apparent in the sequential learning of many tasks. Although GD learning benefits from initial random variations to a large extent, its effects eventually diminishes in a continual learning task. When a series of tasks is posed, the conventional GD learning can use the benefit of random variations only on the first task. For all the subsequent tasks, GD learning is bound to start with weights that are learned by the previous task. If these weights do not form a good starting point for the new task, GD learning will severely suffer. On the other hand, starting from a completely random set of weights each time a new task starts is neither feasible nor desirable. Search with continual injection of random variations may play a substantial role here by amortizing the role of initial randomization over many examples.

We pointed out that representation search through generate and test may play an important role in supervised GD learning and address two issues of it. We noted that GD learning already utilizes a form of search through random initialization, and the random feature generation in representation search may well be seen as an amortized version it. Moreover, testers of feature utility may also be potentially used for solving the problem of catastrophic interference in GD learning, and perhaps in connectionist networks, in general.

We propose to thoroughly investigate the role of representation search in addressing these two issues of supervised GD learning. In all of our experiments next the supervised GD learning is seen as the base system, and representation search operates as an auxiliary to it. The overall input-output map consists of one hidden layer of features. In the following we describe our preliminary studies and findings. Based on these we also discuss what

studies we plan to conduct in future.

We empirically investigate the role of continual feature generation in supervised GD learning. In order to do that, we combine search with GD learning in a particular way and compare it with the standalone GD learning.

Our first experiment is on a regression problem with simulation data similar to that in the previous chapter. We use online backpropagation to minimize the squared error δ^2. Online backpropagation uses a stochastic gradient-descent rule to learn both input and output weights. The output weights are updated using the same LMS rule as in Eq. (11.2). The input weights $\mathbf{v}_{i,k}$ in this case are updated as follows:

$$[\mathbf{v}_{i,k+1}]_j = [\mathbf{v}_{i,k}]_j - \frac{1}{2}\alpha \frac{\partial \delta_k^2}{\partial [\mathbf{v}_{i,k}]_j} = [\mathbf{v}_{i,k}]_j + \alpha \delta_k [\boldsymbol{\theta}_k]_i \frac{\partial [\mathbf{f}_k]_i}{\partial [\mathbf{v}_{i,k}]_j}, \tag{12.1}$$

for $i = 1, \ldots, n$, $j = 1, \ldots, m$.

In order to compare search with GD learning, We tuned the parameters of GD learning method in various ways and obtained the best variant. We experimented with three different kinds of features: logistic functions, hyperbolic tangent functions and LTUs. The GD update of input weights requires computing the derivative of the features. As LTUs are step functions, its partial derivative is zero everywhere except at the threshold. Therefore, the exact GD update for LTUs will not be useful. In order to overcome this problem, we used a modified backpropagation rule for LTUs. Whenever the derivative of a LTU is needed, the derivative of the logistic function is used instead, with the inflection point of the logistic function set at the threshold of the LTU. We tuned both the slope of the sigmoid functions and the initial variance of the input weights. We also used an additional variation of backpropagation. As we mentioned before, the input-weight update of the backpropagation algorithm is proportional to the output weights (see Eq. 12.1). It worsens the problem of catastrophic interference in backpropagation learning, because the update tends to modify the most useful features the fastest due to this update. To alleviate this problem, we use a simple modification where the input-weight update uses only the sign of the output weights:

$$[\mathbf{v}_{i,k+1}]_j = [\mathbf{v}_{i,k}]_j + \alpha \delta_k \operatorname{sign}([\boldsymbol{\theta}_k]_i) \frac{\partial [\mathbf{f}_k]_i}{\partial [\mathbf{v}_{i,k}]_j}, \tag{12.2}$$

where $i = 1, \ldots, n$, $j = 1, \ldots, m$. We refer to it as the *modified gradient update*. Note that, it might be able to counter the worsening of catastrophic interference to some extent, but it cannot eliminate the problem entirely.

When we applied search and GD learning in combination, the GD learning is regarded as the base system. Therefore, for each example, first the backpropagation algorithm updates both the input and the output weights, then the generate and test process is executed. For search, we used the random generator, and the tester based on the trace of output-weight magnitudes. This combination of search and GD learning can be viewed as GD learning with continual injection of randomly generated features. As the total number of features is

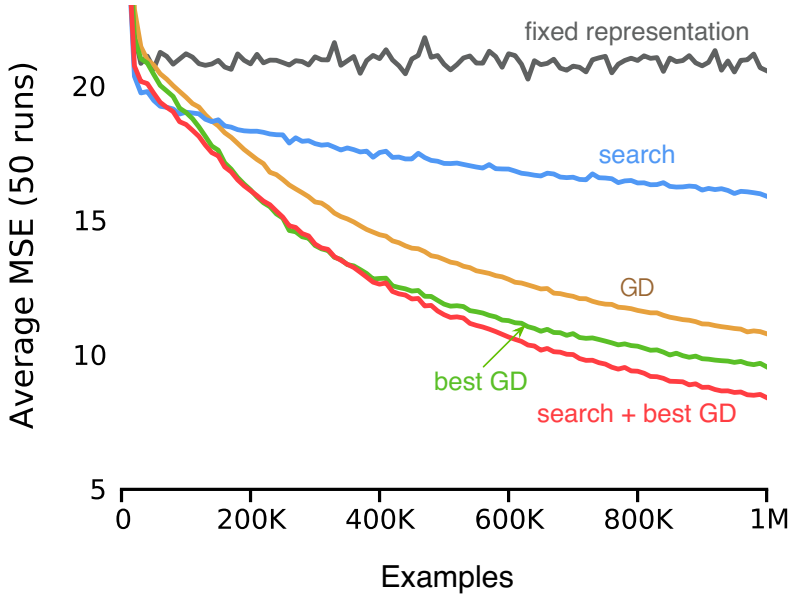

Figure 12.2: Combination of search with supervised gradient-descent (GD) learning performs better than using GD alone.

always constant, the newly generated features always take place of the least useful features so far identified.

For the experiment, we used the same problem as that in the previous chapter, this time with 500 target features and 1000 learnable features. We used more target features than in Section 3 (where we used 20 target features), because when only 20 target features were used to produce the problem, GD learning quickly achieved a low error on the problem and left a little for search to improve on. The results are shown in Figure 12.2. Here, 'GD' refers to the variant of GD where the features are hyperbolic tangent functions, and the modified gradient update is not used. The 'best GD' refers to the variant of GD where the features are LTUs, and the modified gradient update is used. This performed the best among all variants. All the differences in performance are highly statistically significant (the standard errors are smaller than the widths of the lines). The combination of search and the best GD learning reduced the final MSE by 13% more than the best GD alone. This improvement in performance is achieved through a negligible increase to the computational overhead. The extra runtime the combination took was less than 5% of the total runtime taken by the standalone best GD.

This experiment shows that supervised GD learning can indeed benefit from representation search through continual injection of random variations.

12.3 Conclusions

In this chapter, we explored how online representation search interacts with unsupervised learning and backpropagation. We showed that by carefully combining the generate and test method for representation search with unsupervised learning, it is possible to continue harnessing the benefits of unsupervised learning, which is typically utilized for the initialization of the learner. On the other hand, the combination of representation search with backpropagation had been trickier, and the benefits were limiting. We conjecture that as backpropagation can tune the parameters of a learner much faster than a search method can proceed, in single stationary problems it is difficult to improve the performance of backpropagation by adding search. Our study shed some light on how search can play a role in complementing existing representation learning methods, extending their benefits to some novel scenarios, and boosting their performance in some other.

Chapter 13

Conclusions

The core motivation of this thesis has been developing algorithms for large-scale, long-term predictions. We adopted a powerful formulation of this problem: off-policy prediction under the general value function framework, which builds on well-founded concepts such as Markov decision processes and value function estimation. This allows semantic clarity of the prediction problem and utilization of the computational efficiency of temporal-difference (TD) learning algorithms in prediction tasks.

13.1 Summary

We analyzed two core issues with off-policy TD algorithms: they often produce high variance, and they may diverge in some tasks. We made some definitive progress toward overcoming these shortcomings by broadening our understanding of why these issues occur and how they can be prevented. For the issue of high variance, which occurs due to the use of importance sampling and ratios in off-policy algorithms, we took two complementary approaches: utilizing a well-known Monte Carlo estimation method known as weighted importance sampling with function approximation and avoiding an explicit presence of importance sampling ratios altogether. For the issue of instability, we provided a principled approach for analyzing the stability of the TD algorithm and utilized it to produce a stable modification of it.

In the following, we provide a list of key off-policy prediction algorithms developed in this thesis:

- WIS2: A variation of tabular weighted importance sampling that reduces the effect of importance sampling ratios by being aware of state-dependent discounting along the trajectory.
- WIS-LS: A backward-compatible extension of the conventional weighted importance sampling estimator to linear function approximation.

- WIS2-LS: A backward-compatible extension of the WIS2 estimator to linear function approximation.
- WIS-LSTD(λ): A real-time strictly incremental algorithm extending WIS2-LS with bootstrapping and TD updates.
- WIS-TD(λ): An off-policy algorithm with $O(n)$ computational complexity based on weighted importance sampling.
- WIS-GTD(λ): An extension of GTD(λ) based on weighted importance sampling.
- ABQ(ζ): An off-policy algorithm for action-value estimation that avoids an explicit presence of importance sampling ratios by varying the amount of bootstrapping based on state-action pairs in a particular way.
- ETD(λ): An off-policy algorithm with a single tunable parameter and a stability guarantee.

In addition, we introduced a systematic approach to deriving strictly incremental algorithms and producing stable stochastic updates for reinforcement learning. Finally, we developed a computationally efficient method for searching for effective features online, which provides some insight into how knowledge can be curated by a life-long learning agent.

13.2 Future Directions

Off-policy prediction has come a long way through several works including those included in this thesis. Our algorithms are among the state-of-the-art methods for large-scale predictions, and they can be seen as strong solutions to the most notorious issues with off-policy prediction. However, new works always lead to further works and investigations. In the following, we list some of the future directions we deem important:

- **Convergence analysis:** There have been some significant advancements toward our understanding of the convergence of off-policy algorithms. Several recent works (Yu 2010, 2012, 2015, 2016, Maei 2011, Chandrashekar & Bhatnagar 2016) provided convergence proofs for off-policy least-squares algorithms and linear computational complexity algorithms with both simple and two time-scale schemes. These can be adopted to provide convergence proofs for some of the algorithms we introduced in this work.
- **Off-policy prediction with non-linear function approximation:** Theoretical understanding of TD learning with the non-linear function approximation is limited. Nonetheless, many TD algorithms have been developed for non-linear function approximation based on intuitions and heuristics. Although some of these algorithms

have been shown to perform well empirically, the theoretical understanding of these algorithms are still missing, and working toward this can be a significant step for reinforcement learning algorithms.

Having developed algorithms to undertake the issues with off-policy TD learning, the next step would be to utilize them for predictive knowledge representation. Some prior works (Sutton et al. 2011, Modayil 2012, White 2015) have addressed how off-policy TD algorithms can be used for large-scale, long-term predictions. We looked at the problem of curating predictions based on their usefulness by reducing the problem to a supervised representation search problem. However, there are other components of predictive knowledge representation that need to be explored in details, which we list in the following:

- **Grounding knowledge representation to off-policy prediction:** There are some prior works showing how off-policy predictions can be used for knowledge representation. In most of those works, the predictions are used directly as the features. However, some works in predictive state representation hint that a shared feature representation can all be used together to learn some preset predictions. This may also facilitate a substantial transfer of knowledge. These variations need to be explored to determine what would be a preferable scheme for grounding knowledge representation to off-policy prediction.

- **Extending representation search to predictive knowledge representation:** We simplified the problem of curating predictions to the problem of supervised representation search. Therefore, an obvious future step from here is to extend the representation search work to predictive knowledge representation. Both varieties of predictive knowledge representations discussed in the above need to be explored with representation search. If the predictions are used only as the targets but not as features, then our representation search methods readily apply. In this case, the features are of the regular form and can be similar to those we used in our work. On the other hand, if the predictions are used directly as features, then we need to address the problem of how to generate new predictions.

- **Generation of new predictions:** With predictive knowledge representation, a core question, is how to come up with the predictions in the first place that can be used both as the target of learning as well as the elements of the representation. Modayil et al. (2014) hypothesized that, as with animal learning, a knowledge representation could be developed by grounding them to the predictions of the most primitive sensorimotor facts of the agent experience. Starting from such primitive predictions, more complex predictions can be developed through composition and learning sub-goals that are useful for the overall goal of the agent. Moreover, the predictions can be parameterized and learned through gradient descent.

A culmination of our vision of predictive knowledge representation comprises a large number of predictive questions curated through a process of generate and test and corresponding predictive answers learned using an off-policy TD algorithm. Our work is a definitive contribution toward this vision, fleshing out some of the core elements. This brings closer to reality a future where a life-long learning agent can improve and expand continually its knowledge by curating and answering a large number of off-policy predictive questions about the world.

References

Anderson, C. (1993) Q-Learning with hidden-unit restarting. *Advances in Neural Information Processing Systems*, volume 5, S. J. Hanson, J. D. Cowan, and C. L. Giles, eds., Morgan Kaufmann Publishers, San Mateo, CA, pp. 81–88.

Andradóttir, S., Heyman, D. P., Ott, T. J. (1995). On the choice of alternative measures in importance sampling with Markov chains. *Operations Research, 43*(3):509–519.

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

Baird, A. A., Fugelsang, J. A. (2004). The emergence of consequential thought: Evidence from neuroscience. *Phil. Trans. R. Soc. Lond. B 359*:1797–1804.

Barnes, J. M., Underwood, B. J. (1959). "Fate" of first-list associations in transfer theory. *Journal of Experimental Psychology 58*:97–105.

Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H. (2007a). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19.

Bengio, Y., LeCun, Y. (2007b). Scaling learning algorithms towards AI. In DeCoste, D., Bottou, L., Chapelle, O., Weston, J., (Eds.), *Large-Scale Kernel Machines*. MIT Press.

Bengio, Y., Courville, A., Vincent, P. (2012). Unsupervised feature learning and deep learning: A review and new perspectives. *arXiv preprint arXiv*:1206.5538.

Bertsekas, D. P. (1994). A counterexample to temporal-difference learning, *Neural Computation 7*:270–279.

Blum, M., Floyd, R. W., Pratt, V., Rivest, R., Tarjan, R. (1973). Time bounds for selection. Journal of Computer and System Sciences 7: 448–461.

Bousquet, O., Bottou, L. (2008). The tradeoffs of large scale learning. *Advances in neural information processing systems 21*:161–168.

Boyan, J. A., Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. *Advances in Neural Information Processing Systems 1994*, pp. 369–376. MIT Press, Cambridge, MA.

Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning 49*(2):233-246.

Casella, G., Robert, C. P. (1998). Post-processing accept-reject samples: recycling and rescaling. *Journal of Computational and Graphical Statistics, 7*(2):139–157.

Chandrashekar L. and Bhatnagar. S. (2016). A stability criterion for two timescale stochastic approximation schemes, *Automatica*.

Clarkson, P. M. (1993). *Optimal and Adaptive Signal Processing*. CRC press.

Comon, P. (1994). Independent component analysis, a new concept? *Signal Processing 36*(3):287–314.

Cormen, T. H. (2009). *Introduction to Algorithms*. MIT press.

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

David, I. P., Sukhatme, B. V. (1974). On the bias and mean square error of the ratio estimator. *Journal of the American Statistical Association 69*(346):464–466.

Defazio, A., Graepel, T. (2014). A comparison of learning algorithms on the Arcade Learning Environment. *arXiv preprint*, arXiv:1410.8620.

Eicker, F. (1963). Asymptotic normality and consistency of the least squares estimators for families of linear regressions. *The Annals of Mathematical Statistics 34*(2): 447–456.

Eltinge, J. L. (1994). Sufficient conditions for moment approximations for a sample ratio or regression coefficient under simple random sampling. *Sankhyā: The Indian Journal of Statistics*, Series B, 400–414.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research 11*:625–660.

Fahlman, S. E., Lebiere, C. (1990). The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems*, pp. 524–532.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences 3*:128–135.

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

Gordon, G. J. (1995). Stable function approximation in dynamic programming, *Tech. Rep. CMU-CS-95-103*, Carnegie Mellon University.

Graham, R. L., Knuth, D. E., Patashnik, O. (1989). *Concrete Mathematics*. Addison Wesley.

Hammersley, J. M. Handscomb, D. C. (1964). Monte Carlo methods, *Methuen & co. Ltd.*, London, pp. 40,

Harutyunyan, A., Bellemare, M. G., Stepleton, T., Munos, R. (2016). Q (λ) with off-policy corrections. *arXiv preprint* arXiv:1602.04951.

Hastie, T., Tibshirani, R., Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, New York.

Hazan, E., Kale, S. (2008). Extracting certainty from uncertainty: Regret bounded by variation in costs. In *The 21st Annual Conference on Learning Theory*: 57–68.

Hesterberg, T. C. (1988), *Advances in importance sampling*, Ph.D. Dissertation, Statistics Department, Stanford University.

Hinton, G. E., Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science 313*(5786):504–507.

Kleijnen, J. P. C. (1978). Communication: Reply to Fox and Schruben. *Management Science 24*:1772–1774.

Koller, D., Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

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

LeCun, Y. and Bengio, Y. (2007) Scaling Learning Algorithms Towards AI. In Bottou et al. (Eds.) *Large-Scale Kernel Machines*, MIT Press.

Lee, H., Battle, A., Raina, R., Ng, A. Y. (2007). Efficient sparse coding algorithms. *Advances in neural information processing systems 19*.

Ling, R. F. (1974). Comparison of several algorithms for computing sample means and variances. *Journal of the American Statistical Association, 69*(348):859–866.

Liu, J. S. (2001). *Monte Carlo Strategies in Scientific Computing*. Berlin, Springer-Verlag.

Maei, H. R., 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*:91–96. Atlantis Press.

Maei, H. R. (2011). *Gradient Temporal-Difference Learning Algorithms*. PhD thesis, University of Alberta.

Mahmood, A. R., Sutton, R. S., Degris, T., Pilarski, P. M. (2012). Tuning-free step-size adaptation. *Proceedings of the 2012 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Kyoto, Japan, pp. 2121–2124.

Mahmood, A. R., Sutton, R. S. (2013). Representation search through generate and test. In *AAAI Workshop: Learning Rich Representations from Low-Level Sensors*.

Mahmood, A. R., van Hasselt, H., Sutton, R. S. (2014). Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems 27*, Montreal, Canada.

Mahmood, A. R., Sutton, R. S. (2015). Off-policy learning based on weighted importance sampling with linear computational complexity. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*.

Mahmood, A. R., Yu, H., White, M., Sutton, R. S. (2015). Emphatic temporal-difference learning. *European Workshop on Reinforcement Learning 12*,*arXiv preprint* ArXiv:1507.01569.

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

McCloskey, M., Cohen, N. (1989). Catastrophic interference in connectionist networks:

the sequential learning problem, in *The Psychology of Learning and Motivation* (Vol. 24) (Bower, G.H., ed.), pp. 109–164, Academic Press.

McMahan, H. B. (2011). Follow-the-regularized-leader and mirror descent: equivalence theorems and L1 regularization. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*.

Meyer, Jr, C. D., Plemmons, R. J. (1977). Convergent powers of a matrix with applications to iterative methods for singular linear systems. *SIAM Journal on Numerical Analysis 14*(4): 699–705.

Modayil, J., White, A., Sutton, R. S. (2014). Multi-timescale Nexting in a Reinforcement Learning Robot. *Adaptive Behavior 22*(2):146–160.

Munos, R, Stepleton, T, Harutyunyan, A, Bellemare, M. G. (2016). Safe and efficient off-policy reinforcement learning. In *Proceedings of Neural Information Processing Systems*.

Olshausen, B. A., Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by VI? *Vision research*, 37(23), 3311–3325.

Paduraru, C. (2013). *Off-policy evaluation in Markov decision processes*. Doctoral dissertation, PhD thesis, McGill University.

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

Precup, D., Sutton, R. S., Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. In *ICML*:417–424.

Precup, D., Paduraru, C., Koop, A., Sutton, R. S., Singh, S. P. (2005). Off-policy learning with options and recognizers. In *Advances in Neural Information Processing Systems*:1097–1104.

Rafols, E. (2006). *Temporal Abstraction in Temporal-difference Networks* Master's thesis. University of Alberta.

Ratcliff, R. (1990). Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological Review 97*:285–308.

Robins, A. (1995). Catastrophic forgetting, rehearsal, and pseudorehearsal. *Connection Science 7*:123–146.

Robins A. V., Frean M. R. (1998). Local learning algorithms for sequential learning tasks in neural networks. *Journal of Advanced Computational Intelligence 2*(6):107–111.

Roese, N. J. (1997). Counterfactual thinking. *Psychological bulletin 121*(1), 133.

Robert, C. P., and Casella, G., (2004). *Monte Carlo Statistical Methods*, New York, Springer-Verlag.

Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo Method*, New York, Wiley.

Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature 323*:533–536.

Shelton, C. R. (2001). *Importance Sampling for Reinforcement Learning with Multiple Objectives*. Ph.D. thesis, Massachusetts Institute of Technology.

Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference, 90*(2):227–244.

Singh, S. P., Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine learning 22*(1-3):123–158.

Smolensky, P. (1986) Information processing in dynamical systems: foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Volume 1: Foundations, p. 194-281. MIT Press/Bradford Books, Cambridge, MA.

Srebro, N., Sridharan, K., Tewari, A. (2011). On the universality of online mirror descent. In *Advances in Neural Information Processing Systems 24*: 2645–2653.

Sutton, R. S. (1986). Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 823–831.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning 3*(1):9–44.

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

Sutton, R.S., Singh, S.P. (1994). On bias and step size in temporal-difference learning. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pp. 91-96.

Sutton, R. S. (1995). TD models: Modeling the world at a mixture of time scales. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 531–539.

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

Sutton, R. S., Precup, D., Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence 112(1)*:181–211.

Sutton, R. S., Rafols, E. J., Koop, A. (2006). Temporal abstraction in temporal-difference networks. *Advances in Neural Information Processing Systems 18*, 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 Annual International Conference on Machine Learning*:993–1000, ACM.

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

Sutton, R. S., Mahmood, A. R., Precup, D., van Hasselt, H. (2014). A new Q(λ) with interim forward view and Monte Carlo equivalence. In *Proceedings of the 31st International Conference on Machine Learning*, Beijing, China.

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

Thomas, P. S. (2015). *Safe Reinforcement Learning.* PhD Thesis, School of Computer Science, University of Massachusetts Amherst.

Tsitsiklis, J. N., Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning 22*:59–94.

Tsitsiklis, J. N., Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control 42*:674–690.

van Hasselt, H. (2011). *Insights in Reinforcement Learning: formal analysis and empirical evaluation of temporal-difference learning algorithms.* PhD thesis, Universiteit Utrecht.

van Hasselt, H., Mahmood, A. R., Sutton, R. S. (2014). Off-policy TD(λ) with a true online equivalence. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, Quebec City, Canada.

Van Reeken, A. J. (1968). Letters to the editor: Dealing with Neely's algorithms. *Communications of the ACM, 11*(3):149–150.

van Reeken, A. J. (1970). The effect of truncation in statistical computation. *E.I.T. Research Memorandum No. 10, Tilburg Institute of Economics, Tilburg School of Economic.*

van Seijen, H., & Sutton, R. S. (2014). True online TD(λ). In *Proceedings of the 31st International Conference on Machine Learning.* JMLR W&CP 32(1):692–700.

van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., Sutton, R. S. (2016). True online temporal-difference learning. *Journal of Machine Learning Research 17*(145):1–40.

Vapnik, V. (1998) *Statistical Learning Theory.* John Wiley and Sons, Inc., New York.

Varga, R. S. (1962). *Matrix Iterative Analysis.* Englewood Cliffs, NJ: Prentice-Hall.

White, A. (2015). *Developing a Predictive Approach to Knowledge.* PhD thesis, University of Alberta.

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

White, M. (2016). Unifying task specification in reinforcement learning. *arXiv preprint arXiv*:1609.01995.

Yu, H. (2010). Convergence of least squares temporal difference methods under general conditions. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 1207–1214.

Yu, H. (2012). Least squares temporal difference methods: An analysis under general conditions. *SIAM Journal on Control and Optimization 50*(6):3310–3343.

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

Yu, H. (2016). Weak convergence properties of constrained emphatic temporal-difference learning with constant and slowly diminishing stepsize. *Journal of Machine Learning Research 17*(220):1–58.

Zhou, G., Sohn, K., Lee, H. (2012). Online incremental feature learning with denoising autoencoders. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*.