Improving the reliability of reinforcement learning algorithms through biconjugate Bellman errors

by

Andrew Patterson

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

In this thesis, we seek to improve the reliability of reinforcement learning algorithms for nonlinear function approximation. Semi-gradient temporal difference (TD) update rules form the basis of most state-of-the-art value function learning systems despite clear counterexamples proving their potential instability. Gradient TD updates have provable stability under broad conditions, yet significantly underperform semi-gradient approaches on several problems of interest. In this thesis, we present a simple modification to an existing gradient TD method, prove that this method—called TDRC—remains stable, and show empirically that TDRC performs comparatively with semi-gradient approaches. Taking advantage of the connection between Fenchel duality and orthogonal projections, we justify the use of nonlinear value function approximation using gradient TD updates and show that these methods continue to inherit improved reliability over semi-gradient approaches in the nonlinear function approximation setting. We then extend this method to value-based control with neural networks and empirically validate its performance compared to semi-gradient methods. Finally, we propose two novel statistically robust losses—the mean Huber Projected Bellman error and the mean absolute Projected Bellman Error—and derive a family of off-policy gradient TD algorithms to optimize these losses for both prediction and control.

Preface

Most parts of this thesis are based on published works written in collaboration with others.

Chapter 3 is a large subset taken from a work published in the Journal of Machine Learning Research [Patterson et al., 2022b]. My major contributions include ideation and scoping, writing theoretical proofs about the geometry of the objective landscape and the connection to convex conjugates, all experimentation and analysis of results, and a share of the writing of the final manuscript. My advisors, Martha White and Adam White, were heavily involved in the writing of the manuscript as well as ideation and scoping of the project. In addition, Martha wrote the theoretical proofs regarding error bounds.

Chapters 4 and 5 are based on a published paper written with several coauthors [Ghiassian et al., 2020] which introduces one of the primary algorithms of study in this thesis, TDRC. My major contributions to this work include ideation and scoping, inventing the TDRC and QRC algorithms, the theoretical motivation for TDRC, all code used within the project, the empirical design and analysis for the prediction setting, the empirical design for the control setting, and the high-level structure of the convergence proof. My co-first author, Sina Ghiassian, wrote much of the final draft and analyzed the empirical results in the nonlinear control setting. Shivam Garg was responsible for the proof of convergence of the TDRC algorithm.

Finally, Chapter 6 is based on a paper published in the journal Transac-

tions on Pattern Analysis and Machine Intelligence [Patterson et al., 2022a]. My major contributions to this work include ideation and scoping, the theoretical motivation and the connection to biconjugates, all theoretical results and proofs, all empirical design and analysis in the control setting, the empirical design in the prediction setting, and the writing of the final document. My coauthor, Victor Liao, was responsible for the code and empirical analysis in the prediction setting. My advisor, Martha White, played a large role in the writing of the final document as well as the ideation and scoping.

Acknowledgements

This thesis is the product of five years of collaboration with a fantastic group of scientists, researchers, inventors, and careful thinkers. It is impossible to track the many influences of a supportive lab, where just the right word at just the right time can trigger a cascade of new ideas or perspectives. However, there are several people who played a consistent role in my research and growth.

First, I would like to thank my advisor, Martha White, who I have worked with since my undergraduate studies. Martha is a natural teacher and mentor who was willing to train me from barely understanding the introductory RL textbook to the completion of this thesis. Martha has adapted her mentorship over the past eight years, from a technical teacher to a research advisor and finally to helping me carve out a place in our field. Many of my views of mentorship and teaching have been shaped by—and are modelled after—that which I received from Martha.

I would also like to thank Adam White for his ongoing mentorship and collaboration. Adam has been involved in nearly every project that I have worked on to the betterment of the project. Adam is a clear-minded scientist who can see right to the heart of a project and produce a compelling story for the work. My approach to science and research closely model what I have learned from Adam; grumpy skepticism and all!

Finally, I would like to thank my family: my life partner, Katie; my parents; and (most importantly) my two kittens, Moose and Maple. Katie provided support and a fresh perspective as she too developed into a scientist alongside

me. Her insights into statistical best practices and empiricism significantly shaped my approach to science. My parents provided much needed encouragement throughout the degree, allowing me to take my mind off work every December to start the next year refreshed. I was inspired by Moose's curiosity and playfulness and comforted by Maple's warmth and affection.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Building reliable and scalable artificial learning systems is a long-standing goal—and open problem—of reinforcement learning. As we seek to automate increasingly complex and critical tasks, we have an increasing need for systems that have predictable performance. Reliability impacts both the deployment and academic study of artificial learning algorithms, with unreliable systems substantially increasing the cost of both disciplines.

Stability and reliability are both loosely defined in the context of reinforcement learning. This thesis will focus on the consistency of a learning system that is faced with many sources of variation, such as different starting conditions, stochastic streams of data, or variations in deployment scenarios.

Definition 1 Reliability - *A learning system is considered **reliable** in a given environment if the agents produced by that system achieve similar performance, regardless of random initialization of the agents or random influences from the environment.*

Many current state-of-the-art learning systems in reinforcement learning have been shown to be unreliable. Minute changes in configuration parameters can lead to substantially different outcomes and behaviors [Islam et al., 2017, Machado et al., 2018, Henderson et al., 2018, Nagarajan et al., 2019, Engstrom et al., 2019, Patterson et al., 2023], as can differing implementations of the same conceptual architectures. Multiple agents produced from the same learning system can behave so differently, we can wrongly conclude—with statistical significance—that these agents must have come from different learning

systems [Henderson et al., 2018]. Finally, in some cases, taking a learning system that is well-configured for a particular problem and deploying it on a seemingly highly-related problem can lead to abject failure of all agents produced by the system [Machado et al., 2018, Obando-Ceron and Castro, 2021, Patterson et al., 2023].

These examples of unreliability are in no way inevitable, nor are they fundamentally insurmountable. Through careful configuration, the same algorithms that have been described as unreliable have made breakthroughs in solving games designed for humans [Mnih et al., 2013], flying weather balloons in the stratosphere [Bellemare et al., 2020], and controlling cooling systems in data warehouses [Wang et al., 2023]. With the use of pre-existing data, these systems can be pre-trained to avoid much of the instability before deployment [Schwarzer et al., 2021], and this data can be used to carefully tune the configurable parameters of these complex learning systems [Wang et al., 2022]. These approaches, however, often shift the challenge of learning into a challenge of configuration, system engineering acumen, and data curation. That is, we place some of the burden of learning on the system's designer instead of the learning system itself.

The need for consistent learning systems is present both in engineering—where inconsistency can lead to added deployment cost—and science—where inconsistency leads to a lack of reproducibility of claims. It is unsurprising, then, that calls for improving the state of reproducibility in reinforcement learning research have heavily overlapped with empirical demonstrations that many reinforcement learning systems are unreliable [Henderson et al., 2018, Engstrom et al., 2019, Patterson et al., 2023, Obando-Ceron and Castro, 2021, Colas et al., 2018, Agarwal et al., 2021, Pineau et al., 2020]. A common recommendation amongst most—if not all—of these works is to improve the statistical and analytical tooling that we use to understand these unreliable methods, ultimately accepting the premise that these methods need be unreliable. There is, however, an alternative approach: we can strive to produce learning systems that are inherently more reliable. Naturally, these two approaches need not be mutually exclusive; we can improve our analytical tools for unreliable

systems while simultaneously improving the reliability of our systems. In fact, these are likely synergistic.

1.1 Learning value functions

Value functions are at the root of many RL systems in some capacity. Broadly, a value function evaluates a state—or a state-action pair—and approximates the average sum of rewards that will be observed in the future from that state while following a given policy. The value function can be used directly for action selection such as in Q-learning methods [Watkins, 1989, Mnih et al., 2013], as a control variate for variance reduction such as in policy gradient methods [Huang and Jiang, 2020, De Asis and Sutton, 2018], as a component to a model [Schlegel et al., 2021, Sutton et al., 2011, Jaderberg et al., 2016], or as a critic to learn parameterized policies [Bhatnagar et al., 2009, Degris et al., 2012, Liu et al., 2018]. While certainly some value-function free RL methods exist—such as REINFORCE [Williams, 1992]—they are typically constructed from Monte Carlo estimates which are inherently high-variance [Mandel et al., 2014, Thomas and Brunskill, 2016, Jiang and Li, 2016] and can inhibit online, life-long learning [Sutton and Barto, 2018]. These Monte Carlo estimates are typically replaced by model-based [Mannor et al., 2007, Paduraru, 2013] or doubly robust alternatives [Thomas and Brunskill, 2016, Jiang and Li, 2016, Xu et al., 2021], bringing us back to the need for value functions.

Fundamental improvements in value function learning can have a significant impact across nearly all reinforcement learning systems. As such, many algorithms have been developed to improve the methods by which we learn value functions. This includes a variety of variance reduction improvements for off-policy temporal difference algorithms [Precup et al., 2000, Munos et al., 2016, Mahmood et al., 2017]; gradient TD methods with linear function approximation [Sutton et al., 2009, Mahadevan et al., 2014, Liu et al., 2016, Ghiassian et al., 2020] and nonlinear function approximation [Maei et al., 2009]; and algorithms using approximations to the mean squared Bellman error ($\overline{\text{BE}^2}$) [Dai et al., 2017, 2018, Feng et al., 2019].

In online learning, the value function is learned concurrently as the agent interacts with the environment. In its most basic form, online learning occurs synchronously, interwoven between environment interactions. This form of online learning is natural in simulated settings, where the simulator pauses between interactions giving the agent time to update its value function and plan its next action. Another common alternative approach to online learning is to decouple learning and behavior, for instance by retaining a buffer of experience by which the agent updates its value function asynchronously from the environment interaction. In this work, we will consider only simulated environments where the simulator's clock pauses while the agent performs updates.

When the agent's behavior differs from the policy being evaluated, this is known as the off-policy policy evaluation problem. A common application of off-policy learning is when an agent learns many value functions or policies in parallel, for instance to learn option models [Sutton et al., 1999], build predictive representations of state [Littman and Sutton, 2002, Tanner and Sutton, 2005, Sutton et al., 2011, White, 2015, Schlegel et al., 2021], or use auxiliary predictions to improve its state representation [Jaderberg et al., 2016]. In a parallel learning setting, it is natural to estimate the future reward achieved by following each target policy until termination from the states encountered during training—the value of taking excursions from the behavior policy.

In this thesis, I study the problem of online, off-policy policy evaluation and control. Reinforcement learning systems are composed of many complementary components, such as function approximators, models, resampling buffers, exploration methods, and update rules for the value function. Some combinations of these components can lead to unstable learning systems [Sutton and Barto, 2018, van Hasselt et al., 2018, Piché et al., 2023, Zhang et al., 2021, Fan et al., 2020, Fellows et al., 2023]. As an example, current popular update rules are provably unstable based on choices made for other components within the system, namely the function approximator and how samples are generated. This thesis will study update rules which are provably stable under general conditions, allowing greater freedom to openly explore other components of

the learning system without inducing instability or divergence.

1.2 Objective

In this thesis, I aim to improve the reliability of deep reinforcement learning algorithms. Specifically, I address the following question:

> Can we improve the reliability of off-policy, value-based deep reinforcement learning methods without sacrificing performance?

The term *reliability* has no concrete, agreed upon definition in the reinforcement learning literature. In this thesis, when I call a learning method "reliable", I mean learning methods that consistently perform well under random influences; such as stochasticity in the stream of experience, or random initial conditions of the learning system. Implicit to this question is the need to demonstrate that existing methods are *not* reliable according to this definition. The community has begun to establish that existing systems are often reliable only when extensively tuned, and possibly not even then.

The goal of this work is to improve reliability of reinforcement learning methods, it is not necessarily to produce a new algorithm—or class of algorithms—that universally out-perform existing methods. In fact, we would expect that the algorithms proposed in this thesis would typically underperform existing methods for (at least) two reasons. First, if we knew ahead of time that a given problem was easy to solve—say the data is on-policy and the features are rich—then we should expect that an anti-conservative algorithm can perform quite well, while a conservative algorithm will be less sample efficient. Secondly, because we do not want to rely on prior knowledge of the problem, the algorithms proposed in this thesis will first need to use some data to establish the degree to which they can behave conservatively or anti-conservatively. In essence, we will shift some of the work previously handled by extensive algorithm configuration into the algorithm itself, to be adaptively learned by data.

1.3 Contributions

This section describes the three primary contributions of this thesis, relating each contribution back to our stated goal of improving the reliability of deep reinforcement learning algorithms.

Generalized projected Bellman errors (Chapter 3)

This work focuses on the question:

> Which objective should we minimize in order to obtain the best value function estimate for our given parameterized estimator class?

In this chapter, we generalize the mean-squared projected Bellman error ($\overline{\text{PBE}^2}$) through the use its biconjugate. This generalization allows us to reason about the role of the projection in the $\overline{\text{PBE}^2}$ and consider if there are alternative projections that allow our methods to take into consideration the limitations of their function class. The use of biconjugates enables the use of saddlepoint and gradient-based optimization methods to minimize the newly reframed objective. These optimization strategies have provable stability guarantees under broad conditions, which will enable us to derive theoretically sound and stable algorithms for off-policy policy evaluation. One such class of algorithms are the gradient TD family, including GTD2 and TDC [Sutton et al., 2009], which are already known to be stable. The tooling provided by the biconjugate framework allows for novel insights about the performance differential between GTD2, TDC, and semi-gradient methods, where gradient TD methods tend to sacrifice sample efficiency for stability.

Gradient Temporal Difference Learning with Regularized Corrections (Chapter 4)

The previous chapter asks: "*What* should we optimize?" This chapter then asks: "*How* should we optimize it?" In this chapter, we provide an empirical investigation into the performance differential between gradient-based and semi-gradient methods, yielding insights into why gradient TD methods have appeared difficult to use in the past. We then propose a new gradient TD

method, TDRC, and empirically investigate its performance in off-policy prediction, showing that it closes the gap in performance between gradient and semi-gradient methods, while retaining most of the ease-of-use of semi-gradient methods.

Q-Learning with Regularized Corrections (Chapter 5)

We perform an empirical investigation into the performance of QRC, showing that gradient TD-based methods improve reliability without sacrificing performance in nonlinear control. We show that reliability and performance do not depend on the use of target networks for QRC, allowing us to completely remove target networks and avoid tuning this set of configuration parameters. Finally, we discuss two analytical strategies to investigate the reliability of reinforcement learning algorithms and show that QRC is reliable under both strategies, while semi-gradient alternatives are not.

Statistically Robust Bellman Errors (Chapter 6)

This chapter considers two additional objective functions that build on the biconjugate Bellman error presented in Chapter 3. By constraining the space of functions considered by the objective, we can reframe two different statistically robust Bellman errors in a way that is amenable to online optimization. Similar to the TDRC algorithm presented in Chapter 4, we can propose novel families of algorithms to optimize the statistically robust Bellman errors that are provably stable. Unlike the TDRC algorithm—and other prior gradient TD methods—the algorithms proposed in this chapter are robust to high-error states that are rarely visited under a given policy, such as falling off a cliff in the CliffWorld environment while following a near-optimal policy. We motivate both conceptually and empirically that these new losses, and the algorithms that minimize them, can provide increased reliability over prior gradient TD methods in certain environments, and rarely harm performance outside those cases.

Chapter 2

Objectives for Learning Value Functions

In this section, we introduce the notation used throughout this thesis and discuss existing strategies to approximate v_π, the true value function for a given policy. We will define the typical objective whose unique minimizer is v_π, but cannot be estimated from samples. We then discuss several proxy objectives proposed throughout the literature and a couple of known counterexamples which demonstrate conditions for which these objectives are poor proxies.

2.1 Background

We model an agent's interactions with its environment as a Markov Decision Process, $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. At each time-step t, the agent observes state $S_t \in \mathcal{S}$, selects an action $A_t \in \mathcal{A}$ according to policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, transitions to the next state $S_{t+1} \in \mathcal{S}$ according to transition function $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, and receives a scalar reward signal R_{t+1} and discount $\gamma_{t+1} \in [0, 1]$. The discount depends on the transition (state, action and next state), and encodes termination when $\gamma_{t+1} = 0$ [White, 2017].

This cycle of interaction between an agent and its environment continues forever. We assume there are always naturally occurring regularities in the agent's stream of experience. Commonly, an agent might reach a special terminal state and be teleported back to a starting state; such a recurrence is typically called an *episode*. However, there are many other forms of regular,

episodic recurrences in the experience stream such as when the experiment-designer intervenes based on conditions unknown to the agent, and resets the agent to a starting state—often called episode cutoffs or soft-terminations. Further, not all instances of recurrence need to result in disruption of the agent; for instance, an agent that controls the lights in a home automation system and is subject to the natural recurrence of sunlight day over day.

The behavior of an agent is controlled by its policy, typically denoted $\pi(A|S)$ or $b(A|S)$ depending on context. One can evaluate the quality of behavior by tracking the accumulation of (weighted) reward—called the *return*—obtained by that behavior over time. Concretely, we define the return at time t, denoted $G_t \in \mathbb{R}$ as

$$\begin{aligned} G_t &\stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1} R_{t+2} + \gamma_{t+1}\gamma_{t+2} R_{t+3} + \gamma_{t+1}\gamma_{t+2}\gamma_{t+3} R_{t+4} + \dots \\ &= R_{t+1} + \gamma_{t+1} G_{t+1}. \end{aligned}$$

We can estimate the *value* of a behavior using a value function $v_\pi : \mathcal{S} \to \mathbb{R}$ for policy π. The value $v_\pi(s_t)$ describes the expected return an agent would obtain starting from state s_t onward and behaving according to π onward. Formally, we define v_π as

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi \left[G_t \mid S_t = s \right], \text{ for all } s \in \mathcal{S} \tag{2.1}$$

where the expectation operator $\mathbb{E}_\pi[\cdot]$ reflects that the distribution over future actions is given by π.

Often, we wish to estimate $\hat{v}(s) \approx v_\pi \stackrel{\text{def}}{=} \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$ using actions sampled from a distribution other than π. This is known as the *off-policy* learning problem because actions are sampled "off" of, or away from, the target policy, π. We will call the distribution that observed actions are sampled according to the "behavior" policy, b. In the special case that $b = \pi$ for all states and actions, we call this the *on-policy* learning problem.

In this thesis, we will typically assume the agent does not directly observe the state s_t, but instead receives an observation vector $\mathbf{o} \in \mathcal{O} \subset \mathbb{R}^d$ where, $\mathcal{O} = \{\Omega(s) \ \forall \ s \in \mathcal{S}\}$. The mapping $\Omega : \mathcal{S} \to \mathcal{O}$ describes some process by which underlying states are transformed into observations viewed by the

agent; for example, sensors taking measurements of the world. This process is unknown to the agent. The mapping, Ω, may not be injective, multiple states may map to the same observation vector, however the mapping must be surjective, every observation vector in the set $\mathcal{O}$ must be realizable by some state in $\mathcal{S}$. As a result of surjectivity, $\mathcal{O}$ is a countable set whenever $\mathcal{S}$ is a countable set allowing the use of summations over the space instead of integrations. Throughout this thesis, we will present the material assuming a countable $\mathcal{S}$ for ease of exposition. However, in nearly all cases, results trivially hold for continuous, measurable spaces with the appropriate shift from sums to integrals. We will explicitly note where this is not trivial.

Finally, we are interested in problems where the value of each state cannot be stored in a table, for instance because there are infinitely many states. Instead, we will approximate the value with a parameterized function. The approximate value function $\hat{v}(s_t)$ can have arbitrary form, though in this work we require that it is everywhere differentiable with respect to its parameters $w \in \mathbb{R}^d$. We will consider value functions of a fixed feature mapping $\phi : \mathcal{O} \to \Phi$ from observations to features, though for convenience will typically denote these features as vectors $\mathbf{x} \subset \mathbb{R}^d$. Where we can do so unambiguously, we will use the shorthand $\mathbf{x} = \phi(s_t)$ to denote the composition of the observation and feature-generating functions, $\mathbf{x} = (\phi \circ \Omega)(s_t)$.

An important special case is when the approximate value function is linear in the parameters and in features of the state. The value of the state can then be approximated with an inner product: $\hat{v}(s_t) = w^\top \mathbf{x}_t \approx v_\pi(s_t)$. Another typical parameterization for $\hat{v}(s_t)$ is a neural network, where w consists of all the weights in the network. In the case of neural network function approximation, we typically assume that the feature-generating function ϕ is the identity. Alternatively stated, neural network function approximation typically takes the raw observation vectors $\mathbf{o}$ as input.

2.2 Objective functions for learning values

We start by clearly specifying our target objective, the mean-squared value error. We will then show how this objective is difficult to use for online learning using samples. To address this limitation, we will define several proxy objectives that have been used throughout the literature. Much of this section builds on Sutton and Barto [2018, Chapter 11], though rewritten in our own notation. One notable deviation from Sutton and Barto [2018] is the use of the overline to denote "mean squared" as in $\overline{\text{PBE}}$. In this work, we will not always discuss squared errors and so use the overline simply for "mean"—consistent with the statistics literature—and a superscript two for "squared", as in $\overline{\text{PBE}^2}$.

Perhaps the most commonly used metric for evaluating the quality of a value function estimate is the mean squared value error ($\overline{\text{VE}^2}$):

$$\overline{\text{VE}^2}(w) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} d(s)(\hat{v}(s) - v_\pi(s))^2. \tag{2.2}$$

The *mean* denotes that we average the squared residuals, $\hat{v}(s) - v_\pi(s)$, over states with weighting $d(s)$. The usual choice of $d(s)$ is the state visitation frequency under the behavior policy, giving higher weight to states that the agent visits most frequently.

Unfortunately, the $\overline{\text{VE}^2}$ cannot be used as an objective to be optimized with gradient descent. The gradient of the $\overline{\text{VE}^2}$,

$$\nabla_w \overline{\text{VE}^2}(w) = 2 \sum_{s \in \mathcal{S}} d(s) \left(\hat{v}(s) - v_\pi(s)\right) \nabla_w \hat{v}(s)$$

requires that we already know $v_\pi(s)$, yielding a circular dependency: in order to learn v_π, we must already know v_π. We could, however, replace $v_\pi(s)$ with samples, obtaining the mean squared return error ($\overline{\text{RE}^2}$):

$$\overline{\text{RE}^2}(w) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} d(s) \mathbb{E}_\pi \left[\left(\hat{v}(s) - G_t\right)^2 \mid S_t = s \right] \tag{2.3}$$

which uses samples of G_t in place of v_π. Because they share the same gradient,

$\overline{\text{RE}^2}(w)$ and $\overline{\text{VE}^2}(w)$ share the same minimizer

$$\begin{aligned}\nabla\overline{\text{RE}^2}(w) &= \sum_{s\in\mathcal{S}} d(s)\mathbb{E}_\pi\left[(\hat{v}(s) - G_t)\,\nabla\hat{v}(s) \mid S_t = s\right] \\ &= 2\sum_{s\in\mathcal{S}} d(s)\left(\hat{v}(s) - \mathbb{E}_\pi\left[G_t \mid S_t = s\right]\right)\nabla\hat{v}(s) \\ &= 2\sum_{s\in\mathcal{S}} d(s)\left(\hat{v}(s) - v_\pi(s)\right)\nabla\hat{v}(s) = \nabla\overline{\text{VE}^2}(w).\end{aligned}$$

In practice, the $\overline{\text{RE}^2}$ is rarely used because it requires obtaining samples of entire returns. This restricts algorithms minimizing the $\overline{\text{RE}^2}$ to offline updates, updating only at the end of episodes. Samples of G_t are also notoriously high variance [Thomas and Brunskill, 2016, Jiang and Li, 2016], a problem which is amplified in the off-policy setting. Despite these limitations, the fact that the $\overline{\text{RE}^2}$ provides an unbiased estimator for $\arg\min_w \overline{\text{VE}^2}(w)$ makes this an appealing proxy objective to pursue.

A natural alternative is to take advantage of the fact that the fixed-point of the Bellman operator ($\mathcal{T}v$) is unique and is the true value function, v_π. That is, the recursive relationship $v = \mathcal{T}v$ is uniquely satisfied by $v = v_\pi$. We define the Bellman residual as

$$\Delta(s) \stackrel{\text{def}}{=} \mathcal{T}\hat{v}(s) - \hat{v}(s)$$

where $\mathcal{T}\hat{v}(s_t) = \mathbb{E}_\pi\left[R_{t+1} + \gamma_{t+1}\hat{v}(S_{t+1}) \mid S = s\right]$ is the Bellman operator. The mean squared Bellman error then squares these residuals and takes the weighted average over states,

$$\overline{\text{BE}^2}(w) \stackrel{\text{def}}{=} \sum_{s\in\mathcal{S}} d(s)\Delta(s)^2 = \sum_{s\in\mathcal{S}} d(s)\mathbb{E}_\pi\left[\delta(w) \mid S = s\right]^2 \tag{2.4}$$

where we write $\delta(w)$ to be explicit that this is the TD error for the parameters w. The $\overline{\text{BE}^2}$ is minimized when $\hat{v} = \mathcal{T}\hat{v}$, implying that $\hat{v} = v_\pi$. However, in the function approximation setting the minimizer of the $\overline{\text{BE}^2}$ may not be realizable, resulting in some residual errors. The resulting value function trades off between accurately estimating v_π and providing a useful bootstrapping target for every state.

The final commonly used proxy objective is the mean squared *projected* Bellman error; the $\overline{\text{PBE}^2}$. Similar to the $\overline{\text{BE}^2}$, the $\overline{\text{PBE}^2}$ minimizes an average

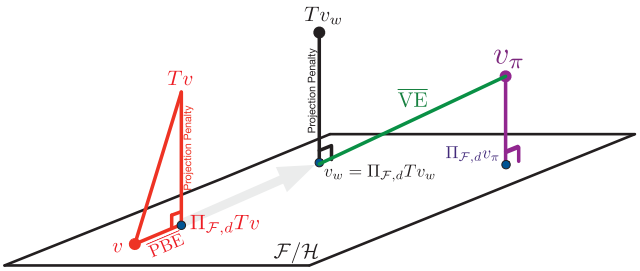

Figure 2.1: The visualization above characterizes the true v_π, the $\overline{\text{PBE}^2}$ solution, and how projections operate on successive approximations. Assume the estimate of v_π starts from $\mathbf{v}$ in red. The Bellman operator pushes the value estimate out of the space of representable functions represented by the plane. The projection brings the approximation back down to the nearest representable function on the plane. This process is repeated over and over until the value estimates converge to the blue dot at the base of the black line. Subsequent updates push the approximation to v_π out of the space of representable functions and the projection back onto the plane. The true value in this case is outside $\mathcal{V}$, with the $\overline{\text{VE}^2}$ being the distance between the $\mathbf{v}$ at $\overline{\text{PBE}^2} = 0$ and v_π. Note the projection of v_π onto $\mathcal{V}$ need not be equal to $\overline{\text{PBE}^2}$ solution.

Bellman residual. Unlike the $\overline{\text{BE}^2}$, the $\overline{\text{PBE}^2}$ first projects the Bellman residual onto the space spanned by the features, $\mathbf{X}$, eliminating sources of error which cannot be represented. The $\overline{\text{PBE}^2}$ is defined as

$$\overline{\text{PBE}^2}(w) \stackrel{\text{def}}{=} \|\hat{v} - \Pi_{\mathcal{V}} \mathcal{T} \hat{v}\|_d^2 \tag{2.5}$$

where $\Pi_{\mathcal{V}}$ is the orthogonal projection onto the vector space $\mathcal{V} = \text{span}(\mathbf{X})$. Note that orthogonality in this Hilbert space is defined by the weighted inner-product $< x, y >_d = x^\top D y$ where $D = \text{diag}(d)$.

There has been much discussion, formal and informal, about using the $\overline{\text{BE}^2}$ versus the $\overline{\text{PBE}^2}$. There is a clear relationship between the $\overline{\text{BE}^2}$ and the $\overline{\text{PBE}^2}$ through the triangle inequality [Scherrer, 2010] when the projection $\Pi_{\mathcal{V}}$ is orthogonal

$$\underbrace{\|v - \mathcal{T} v\|_d^2}_{\overline{\text{BE}^2}} = \underbrace{\|v - \Pi_{\mathcal{V}} \mathcal{T} v\|_d^2}_{\overline{\text{PBE}^2}} + \underbrace{\|\mathcal{T} v - \Pi_{\mathcal{V}} \mathcal{T} v\|_d^2}_{\text{Projection Penalty}}. \tag{2.6}$$

This penalty causes the $\overline{\text{BE}^2}$ to prefer value estimates for which the projection does not have a large impact near the solution. The $\overline{\text{PBE}^2}$ can find a fixed

point where applying the Bellman operator $\mathcal{T}v$ moves far outside the space of representable functions, as long as the projection back into the space stays at v.

2.3 Known challenges with existing objectives

Despite the potential utility of the $\overline{\text{BE}^2}$, it has not been widely used due to difficulties in optimizing this objective without a model. The $\overline{\text{BE}^2}$ is difficult to optimize because of the well-known double sampling problem for the gradient. To see why, consider the gradient

$$\begin{aligned}
\nabla_w \overline{\text{BE}^2}(w) &= \sum_{s \in \mathcal{S}} d(s) \nabla_w \Delta(s)^2 \\
&= 2 \sum_{s \in \mathcal{S}} d(s) \Delta(s) \mathbb{E}_\pi \left[\nabla_w \delta(w) \mid S = s \right] \\
&= 2 \sum_{s \in \mathcal{S}} d(s) \Delta(s) \mathbb{E}_\pi \left[\gamma \nabla_w \hat{v}(S') - \nabla_w \hat{v}(s) \mid S = s \right] \\
&= 2 \sum_{s \in \mathcal{S}} d(s) \mathbb{E}_\pi \left[\delta(w) \mid S = s \right] \mathbb{E}_\pi \left[\gamma \nabla_w \hat{v}(S') - \nabla_w \hat{v}(s) \mid S = s \right].
\end{aligned}$$

To estimate this gradient for a given $S = s$, we need two independent samples of the next state and reward. We use the first to get a sample $\delta(w)$ and the second to get a sample of $\gamma \nabla_w \hat{v}(S') - \nabla_w \hat{v}(s)$. The product of these two samples gives an unbiased sample of the product of the expectations. If we instead only used one sample, we would erroneously obtain a sample of $\mathbb{E}_\pi \left[\delta(w)(\gamma \nabla_w \hat{v}(S') - \nabla_w \hat{v}(s)) \mid S = s \right]$ which is the gradient of a mean squared TD error,

$$\overline{\text{TDE}^2}(w) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} d(s) \mathbb{E}_\pi \left[\delta(w)^2 \mid S = s \right]. \tag{2.7}$$

It has been well-established that solutions to the $\overline{\text{TDE}^2}$ provide poor approximations to v_π [Sutton and Barto, 2018, Patterson et al., 2022b].

The linear $\overline{\text{PBE}^2}$, on the other hand, is practical to optimize under linear function approximation; algorithms such as TD converge to the minimum of the $\overline{\text{PBE}^2}$ when they converge [Sutton et al., 2009, Maei, 2011]. Other algorithms, such as the gradient TD family of methods [Antos et al., 2008, Sutton et al., 2009], have guaranteed convergence to the minimum of the $\overline{\text{PBE}^2}$

under general conditions. Unfortunately, the $\overline{\text{PBE}^2}$ is hard to optimize for the nonlinear setting due to the linear projection. Prior attempts to extend GTD to the nonlinear $\overline{\text{PBE}^2}$ [Maei et al., 2009] resulted in an algorithm that estimates a local linearization of the projection, resulting in a Hessian-vector product.

2.4 Algorithms

The most widely used value function learning algorithm is also one of the oldest: temporal difference (TD) learning. The continued popularity of TD stems from both (a) the algorithm's simplicity and empirical performance, and (b) the lack of technical tools required to improve it. TD, however, does not follow the gradient of any known objective function [Baird, 1995, Antos et al., 2008], and without a clear objective for TD, it is difficult to characterize its behavior. Related residual gradient algorithms directly optimize a known objective, the mean squared Bellman error ($\overline{\text{BE}^2}$), but suffer from the double sampling problem [Baird, 1995, Scherrer, 2010] resulting in a biased proxy-objective instead: the mean squared temporal difference error ($\overline{\text{TDE}^2}$). Without a strategy to optimize the $\overline{\text{BE}^2}$ in the absence of a simulator, it was difficult to pursue the $\overline{\text{BE}^2}$ as an alternative.

Several alternative strategies have been proposed to minimize the $\overline{\text{BE}^2}$. For instance, one could minimize the $\overline{\text{TDE}^2}$, while simultaneously estimating a bias correction term [Antos et al., 2008]. Using this estimate, one can make bias-corrections to offset the error induced by using statistically dependent samples. Alternatively, one could change the update used in the residual gradient methods in order to use independent samples of a quantity that *can* be sampled repeatedly given a sufficient large buffer [Feng et al., 2019]. A final alternative strategy is to estimate part of the gradient update, replacing the bias induced by dependent samples with an approximation bias that we can more easily control [Sutton et al., 2009, Dai et al., 2018]. The gradient TD family of algorithms—including TDC and GTD2—was derived using this last approach.

In parallel, several works began improving the stability of semi-gradient TD methods, applying mitigation strategies to avoid stability issues with the learning system. These mitigations include delaying the propagation of information when using bootstrapped estimates with target networks [Mnih et al., 2015, Fan et al., 2020], clipping rewards [Hessel et al., 2018b], clipping errors and gradients [Mnih et al., 2015, Van Hasselt et al., 2016], or careful manipulation of the reward function [Brockman et al., 2016, Young and Tian, 2019]. These mitigation strategies attempt to avoid instability in the learning system by careful problem-specific tuning of their various configuration parameters.

Although the use of target networks has become near-ubiquitous in modern learning systems, it is unclear what impact their use has on the system. Target networks can provide stability to the learning system by reducing the variance of the learning target [Fan et al., 2020, Piché et al., 2023, Fellows et al., 2023], though the impact of this variance reduction on convergence remains unclear. Additionally, by reducing the rate that new information is used in bootstrapping, target networks tend to harm sample efficiency [Kim et al., 2019, Hernandez-Garcia and Sutton, 2019, Piché et al., 2023]. Finally, many learning systems have been shown to be sensitive to the hyperparameters used to configure the target network, with frequent synchronization leading toward stability issues and infrequent synchronization significantly slowing learning [Obando-Ceron and Castro, 2021, Piché et al., 2023].

2.5 Summary

In this chapter, we briefly covered the necessary background to provide the context for the work in the rest of this document. We defined the reinforcement learning problem setting and established notation. Perhaps the most important discussion is our definition of states, observations, and features, which will play an important role later as we define our objective functions. Previous work, including prior publication of this work, have typically used observations and features interchangeably. However, in this thesis we will consider observations as coming from the environment and features as internal to

the agent. The agent can freely use both observations and features constructed from those observations to inform its value function estimates, but the agent cannot take advantage of the underlying states as these are unobserved.

Finally, this chapter covers previous objective functions used for value function learning. Much of this chapter closely follows Chapter 11 of Sutton and Barto [2018]. We highlight minor differences in terminology as compared to prior work and highlight the drawbacks with existing objective functions. The next chapter will unify two of the present objective functions, the $\overline{\text{BE}^2}$ and the $\overline{\text{PBE}^2}$ and use this unification to address some drawbacks with each.

Chapter 3

Generalized Projected Bellman Errors

In this chapter, we develop a framework for (bi)conjugate Bellman errors lightly extending the work done by Dai et al. [2017]. The use of biconjugates provides a general framework that unifies many prior proxy objectives, such as the $\overline{\text{BE}^2}$ and the $\overline{\text{PBE}^2}$. The biconjugate objectives avoid the problem of double sampling, allowing us to later construct gradient-based algorithms which minimize these objectives.

Using the equality of the biconjugate with the original function, allows the lossless rewrite the $\overline{\text{BE}^2}$ in a form that is amenable to online learning. When the maximizer h^* is not feasible, or when there is approximation error in estimating the maximizer, then we obtain an approximation to the $\overline{\text{BE}^2}$. We show in Section 3.2 that the approximation due to representability leads exactly to the $\overline{\text{PBE}^2}$ under certain conditions on h.

3.1 Biconjugate Bellman Errors

For a real-valued function $f : \mathbb{R} \to \mathbb{R}$, the conjugate is $f^*(h) \stackrel{\text{def}}{=} \sup_{x \in \mathbb{R}} xh - f(x)$. This function f^* also has a conjugate, f^{**}, which is called the biconjugate of f. Further, for any function f that is proper, convex, and lower semi-continuous, the biconjugate $f^{**}(x) = f(x)$ for all x by the Fenchel-Moreau theorem [Fenchel, 1949, Moreau, 1970]. In the unique special case of the square function, the conjugate and biconjugate are the same; that is $f(x) =$

$f^*(x) = f^{**}(x) = x^2$. Throughout this section, we will take advantage of the fact that the conjugate and biconjugate of the square function are the same—allowing us to reformulate the Bellman error using only the conjugate. In later chapters, we will require the full biconjugate for our formulation.

Let $\mathcal{V}$ be the space of parameterized value functions and $\mathcal{F}$ the space of all functions mapping $\mathcal{O} \to \mathbb{R}$. To reformulate the $\overline{\text{BE}^2}$, we will use the fact that the conjugate of the square function is $y^2 = \max_{h \in \mathbb{R}} 2yh - h^2$ and the fact that the maximum can be brought outside the sum (interchangeability), as long as a different scalar h can be chosen for each state s. Then the $\overline{\text{BE}^2}$ can be rewritten as

$$\begin{aligned}
\overline{\text{BE}^2}(w) &= \sum_{s \in \mathcal{S}} d(s) \Delta(s)^2 && \triangleright \text{ Let } \Delta(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi \left[\delta(w) \mid S = s\right] \\
&= \sum_{s \in \mathcal{S}} d(s) \max_{h \in \mathbb{R}} \left(2\Delta(s)h - h^2\right) && \triangleright \text{ conjugate} \\
&= \max_{h \in \mathcal{F}} \sum_{s \in \mathcal{S}} d(s) \left(2\Delta(s)h(s) - h(s)^2\right) && \triangleright \text{ interchangeability.}
\end{aligned}$$

It is clear to see that the conjugate of the square function attains its maximum when the inner variable $h = y$. Therefore, using the optimal $h^*(s) = \Delta(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi \left[\delta(w) \mid S = s\right]$,

$$\begin{aligned}
&\sum_{s \in \mathcal{S}} d(s) \left(2\Delta(s)h^*(s) - h^*(s)^2\right) && \triangleright \text{ Conjugate at } h^*(s) \\
&\quad = \sum_{s \in \mathcal{S}} d(s) \left(2\Delta(s)^2 - \Delta(s)^2\right) && \triangleright \text{ Substitute } h^*(s) = \Delta(s) \\
&\quad = \sum_{s \in \mathcal{S}} d(s) \Delta(s)^2 \\
&\quad = \overline{\text{BE}^2}
\end{aligned}$$

it is clear that for $h(s) = h^*(s)$ this reformulation is simply a rewriting of the $\overline{\text{BE}^2}$. Dai et al. [2018] use a similar reformulation of the $\overline{\text{BE}^2}$, except instead of using $h(s)$ to track the Bellman residual, they track the one-step Bellman operator, $h(s) \approx \mathbb{E}_\pi \left[\mathcal{T}\hat{v}(s) \mid S = s\right]$. As a result, their reformulation is equivalent up to an affine transformation resulting in slightly different gradients for each $\hat{v}(s)$ and $h(s)$ than our reformulation. The algorithm proposed in Dai et al. [2018], SBEED, then will be highly related to the algorithms proposed in

this thesis, typically differing only in minute design decisions such as tracking Bellman residuals instead of Bellman backups.

Empirically, the choice to estimate Bellman residuals appears to have major performance implications [Patterson et al., 2022b]. With the Bellman residual, we can take advantage of our prior knowledge: samples of the Bellman residual (i.e. TD errors) generally have zero mean and are of the same magnitude as one-step rewards. With bootstrapped values, we often have no *a priori* estimate of their mean or their magnitude. Using this prior knowledge allows designing smarter update rules and improved modeling decisions.

With biconjugates, we replace the requirement for two independent samples with a single sample and a regressor. Alternatively, one could draw two dependent samples then transform the estimate into the $\overline{\text{BE}^2}$ by cancelling out the bias term with a secondary estimator [Antos et al., 2008]. We can rewrite the $\overline{\text{BE}^2}$ as a function of both w and h, $\overline{\text{BE}^2}(w) = \max_{h \in \mathcal{F}} L(w, h)$. Then the strategy of Antos et al. [2008] uses $L(w, h) = \overline{\text{TDE}^2}(w) - \|h(w) - \mathcal{T}\hat{v}\|_d^2$. Note that, similar to SBEED, the secondary function here tracks the one-step Bellman update.

A final alternative approach is to use a non-parametric estimator for h. For example, a reservoir of transitions can be stored and $\mathbb{E}_\pi [\delta_t \mid S_t = s]$ is approximated using a weighted average over δ_t in the buffer, where the weighting is proportional to similarity between that state and s. This is the strategy taken by the Kernel $\overline{\text{BE}^2}$ [Feng et al., 2019], precisely to reduce bias in h and so better approximate the $\overline{\text{BE}^2}$. When learning online, this non-parametric approach is less practical; either a large buffer needs to be maintained, or a sufficient set of representative transitions identified and stored.

3.2 A generalized projected Bellman error

As highlighted in [Sutton and Barto, 2018, Chapter 11.6], the $\overline{\text{BE}^2}$ is not learnable. There, an entity is considered learnable if it can be uniquely defined given observable data. Given two identical data streams, the best-in-class value estimate $v \in \mathcal{V}$ for both data streams should likewise be identical. For the $\overline{\text{BE}^2}$,

this is not the case. If each stream is generated by a different unobservable process, then the best-in-class v is different. The agent, however, has no way to know this.

A simple example pair of MDPs succinctly highlight the problem. The first is a two state MDP where the agent deterministically transitions from state A to $\tilde{B}$ and stays in state $\tilde{B}$ with 50% probability, or returns to A. The second is a three state MDP, but two of the states are aliased and indistinguishable with the given observations. The second MDP shares the same structure, from state A the agent deterministically transitions to *one of* the aliased states (collectively, state $\tilde{B} = \{B, B'\}$). Then with 50% probability, the agent transitions between the two aliased states within $\tilde{B}$ or transitions from $\tilde{B}$ back to A. Each state has equal probability of occurrence across MDPs, and the transition dynamics between observable states A and $\tilde{B}$ are equivalent.

Although both MDPs produce identical streams of experience, the solution to the $\overline{\text{BE}^2}$ differs for each MDP. Yet, if the agent cannot identify which MDP is producing its experience, then the agent cannot identify the $\overline{\text{BE}^2}$ minimizer for its given MDP. That is, the issue of learnability is an issue of identifiability—the $\overline{\text{BE}^2}$ does not emit a unique minimizer conditioned on the stream of observed data. According to the proxy objective, the $\overline{\text{BE}^2}$, each minimizer is equally good. According to our true objective, the $\overline{\text{VE}^2}$, only one of these minimizers is the best. The other potential minimizers can be notably worse. Determining the unique minimizer of the $\overline{\text{BE}^2}$ requires unobservable information; namely the true underlying states. Note that despite the partial observability, both streams of experience are valid Markov processes; this issue of identifiability need not result from breaking the Markov assumption.

In the above example, partial observability means that the agent can only observe a part of the state for learning $\hat{v}$. However, the biconjugate formulation of the $\overline{\text{BE}^2}$ requires that the agent observe the whole state for evaluating $h(s)$. This suggests a contradiction: the agent can learn a limited value function using only partial observations of the state, but may propose a sequence of loss functions which can observe the entirety of the state. Naturally, the input-space for h should be similarly restricted to only observable information. We

can define the set of such functions:

$$\mathcal{H} \stackrel{\text{def}}{=} \{h = f \circ \mathbf{x} \mid \text{ where } f \text{ is any function on the space produced by } \mathbf{x}\}$$

where $\mathbf{x} : \mathcal{O} \to \mathbf{X}$ produces the same features used for $\hat{v}$. We call the resulting $\overline{\text{BE}^2}$ an *Identifiable* $\overline{\text{BE}^2}$, written as:

$$\text{Identifable BE}(w) \stackrel{\text{def}}{=} \max_{h \in \mathcal{H}} \mathbb{E}\left[2\Delta(s)h(s) - h(s)^2\right].$$

Notice that $\mathcal{H} \subseteq \mathcal{F}$, and so the solution to the Identifable BE may be different from the solution to the $\overline{\text{BE}^2}$. In particular, we know the Identifable $\text{BE}(w) \leq \overline{\text{BE}^2}(w)$ because the inner maximization is more constrained.

Optimizing this Identifable BE, however, is not typically possible as we will often not be able to perfectly represent *any* function of the features. Instead, we are typically constrained to a class of parameterized functions of the features—for instance the set of all linear functions, or the set of all neural networks of a given architecture. Each of these function classes form a subset $\mathcal{H}_\theta \subseteq \mathcal{H}$, with θ the vector of parameters defining the vector space $\mathcal{H}_\theta$. This subset further constrains the proxy objective.

To analyze the impact of parameterized h, we first define an orthogonal projection operator $\Pi_{\mathcal{H}_\theta,d}$ which projects any vector $u \in \mathbb{R}^{|\mathcal{S}|}$ onto convex subset $\mathcal{H}_\theta \subseteq \mathbb{R}^{|\mathcal{S}|}$ under state weighting d:

$$\Pi_{\mathcal{H}_\theta,d} u \stackrel{\text{def}}{=} \arg\min_{h \in \mathcal{H}_\theta} \|u - h\|_d. \tag{3.1}$$

Note the weighting d here is critical for defining an orthogonal projection in the Hilbert space weighted by d. For all projections, we will assume a weighted Hilbert space and will drop the subscript d to simplify notation.

We define the generalized $\overline{\text{PBE}^2}$ as

$$\overline{\text{PBE}^2}(w) \stackrel{\text{def}}{=} \|\Pi_{\mathcal{H}_\theta}(\mathcal{T}\hat{v} - \hat{v})\|_d^2 \tag{3.2}$$

where each choice of $\mathcal{H}_\theta$ results in different projection operators. This view provides some intuition about the role of approximating h—different projections will trade off Bellman error across different states. If the function approximator has high Bellman error in a given state, but that error is projected to zero,

then no further approximation resources will be used for that state. In the case of full observability and the set for h is the set of all functions, then no errors are projected and the values are learned to minimize the Bellman error. If $\mathcal{H}_\theta = \mathcal{V}$, the same space is used to represent h and v, then we obtain the projection originally used for the $\overline{\text{PBE}^2}$.

We now show the connection between this $\overline{\text{PBE}^2}$ and the $\overline{\text{BE}^2}$. In the finite state setting, we have a vector $u \in \mathbb{R}^{|\mathcal{S}|}$ composed of entries $u_s = \Delta(s)$. Because $\Pi_{\mathcal{H}_\theta,d}$ is an orthogonal projection, $u = \Pi_{\mathcal{H}_\theta,d} u + \tilde{u} = h + \tilde{u}$, where $h = \Pi_{\mathcal{H}_\theta,d} u$ and $\tilde{u}$ is the component in u that is orthogonal in the weighted space: $h^\top D \tilde{u} = 0$ for $D \stackrel{\text{def}}{=} \text{diag}(d)$. Then we can write the conjugate form for the $\overline{\text{BE}^2}$, now with restricted $\mathcal{H}_\theta \subset \mathcal{F}$

$$
\begin{aligned}
&\max_{h \in \mathcal{H}_\theta} \sum_{s \in \mathcal{S}} d(s) \left(2\Delta(s)h(s) - h(s)^2\right) \\
&\quad = \max_{h \in \mathcal{H}_\theta} \sum_{s \in \mathcal{S}} d(s) \left(2u(s)h(s) - h(s)^2\right) && \triangleright \text{ rewriting } u(s) = \Delta(s) \\
&\quad = \sum_{s \in \mathcal{S}} d(s) \left(2u(s)h(s) - h(s)^2\right) && \triangleright \text{ where } h = \Pi_{\mathcal{H}_\theta,d} u \\
&\quad = \sum_{s \in \mathcal{S}} d(s) \left(2[h(s) + \tilde{u}(s)]h(s) - h(s)^2\right) && \triangleright \text{ because } u(s) = h(s) + \tilde{u}(s) \\
&\quad = \sum_{s \in \mathcal{S}} d(s) \left(2h(s)^2 - h(s)^2\right) + 2 \sum_{s \in \mathcal{S}} d(s)\tilde{u}(s)h(s) \\
&\quad = \sum_{s \in \mathcal{S}} d(s) h(s)^2 + 2 \sum_{s \in \mathcal{S}} d(s)\tilde{u}(s)h(s) \\
&\quad = \sum_{s \in \mathcal{S}} d(s) h(s)^2 && \triangleright \text{ where } \sum_{s \in \mathcal{S}} d(s)\tilde{u}(s)h(s) = 0 \text{ because} \\
&\quad = \|\Pi_{\mathcal{H}_\theta,d}(\mathcal{T}\hat{v} - \hat{v})\|_d^2 && \quad h \text{ is orthogonal to } \tilde{u}, \text{ under weighting } d \\
&\quad = \overline{\text{PBE}^2}(w)
\end{aligned}
$$

By restricting $\mathcal{H}_\theta$ to be a Hilbert space, we can ensure that the projection operator $\Pi_{\mathcal{H}_\theta,d}$ is an orthogonal projection. Naturally, this assumption is easily satisfied by linear functions with a fixed basis and with bounded weights. Many classes of neural networks have been shown to be expressible as reproducing kernel Hilbert spaces (see Bietti and Mairal [2019] for a nice overview), including neural networks with ReLU activations as are commonly used in RL. Therefore, this is not an overly restrictive assumption.

There are many feasible choices for estimating h. Likely the simplest is to use the same approximator for h as for the values. For example, this might mean that h and $\hat{v}$ use the same features such as two heads on a shared neural network. However, we could feasibly consider a much bigger class for h, because h is only used during training, not prediction. Because, we typically want $\hat{v}$ to be efficient to query, we might use a more compact parameterized function approximator for $\hat{v}$ while h uses a more computationally costly function approximator, updated asynchronously between agent-environment interaction steps.

Connecting to previous objectives. In this section we show how the generalized $\overline{\text{PBE}^2}$ lets us express the linear $\overline{\text{PBE}^2}$ and nonlinear $\overline{\text{PBE}^2}$ by selecting different sets, $\mathcal{H}$. First let us consider the linear $\overline{\text{PBE}^2}$. To connect the generalized $\overline{\text{PBE}^2}$ to the linear $\overline{\text{PBE}^2}$, we start by restating the original construction of the linear $\overline{\text{PBE}^2}$ [Sutton et al., 2009], then show the connection to the saddlepoint formulations developed for the linear $\overline{\text{PBE}^2}$ [Mahadevan et al., 2014, Liu et al., 2016, Touati et al., 2018]. From there, it is straightforward to show the connection between the saddlepoint formulation and our biconjugate formulation—which itself forms a saddlepoint [Dai et al., 2017, 2018].

Define the set of all linear functions on $\mathbf{x}$ as $\mathcal{L} = \{f : \mathcal{S} \to \mathbb{R} : f(s) = \mathbf{x}(s)^\top w, w \in \mathbb{R}^d\}$. In the linear setting where $\mathcal{V} = \mathcal{H} = \mathcal{L}$, then we can define the $\overline{\text{PBE}^2}$

$$\begin{aligned}
\overline{\text{PBE}^2}(w) &= \|\hat{v} - \Pi_{\mathcal{L}}\mathcal{T}\hat{v}\|_d^2 \\
&= \|\mathbf{b} - \mathbf{A}w\|_{\mathbf{C}^{-1}}^2 \\
&= \mathbb{E}_d\left[\delta\mathbf{x}\right]^\top \mathbb{E}_d\left[\mathbf{x}\mathbf{x}^\top\right]^{-1} \mathbb{E}_d\left[\delta\mathbf{x}\right]
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{A} &\doteq \mathbb{E}_d\left[\mathbf{x}(\mathbf{x} - \gamma\mathbf{x}')^\top\right] \\
\mathbf{b} &\doteq \mathbb{E}_d\left[R\mathbf{x}\right] \\
\mathbf{C} &\doteq \mathbb{E}_d\left[\mathbf{x}\mathbf{x}^\top\right]
\end{aligned}$$

and the notation $\mathbb{E}_d[Y] = \sum_{s\in\mathcal{S}} d(s)Y(s)$ denotes the expectation of Y over states, distributed according to state distribution d.

Rewriting this linear $\overline{\text{PBE}^2}$ using biconjugates is straightforward. The biconjugate for the two-norm is

$$\tfrac{1}{2}\|\mathbf{y}\|_{\mathbf{C}^{-1}} = \max_{\theta} \mathbf{y}^\top\theta - \tfrac{1}{2}\|\theta\|^2_{\mathbf{C}},$$

with optimal $\theta = \mathbf{C}^{-1}\mathbf{y}$. Correspondingly, we get

$$\tfrac{1}{2}\|\mathbf{b} - \mathbf{A}w\|^2_{\mathbf{C}^{-1}} = \max_{\theta\in\mathcal{L}} (\mathbf{b} - \mathbf{A}w)^\top\theta - \tfrac{1}{2}\|\theta\|^2_{\mathbf{C}}.$$

And finally, the inner optimization of the biconjugate becomes

$$\theta^* = \arg\min_{\theta\in\mathcal{L}} \mathbb{E}_d\left[(\Delta(s) - \mathbf{x}(s)^\top\theta)^2\right],$$

which is a simple least squares optimization. The solution to which is

$$\begin{aligned}\theta^* &= \mathbf{C}^{-1}(\mathbf{b} - \mathbf{A}w)\\ &= \mathbb{E}_d\left[\mathbf{x}\mathbf{x}^\top\right]^{-1}\mathbb{E}_d\left[\mathbf{x}\delta\right].\end{aligned}$$

This result is alluded to in the connection between the NEU and the $\overline{\text{KBE}^2}$ in Feng et al. [2019, Corollary 3.5], but not explicitly shown.

This connection also exists with the nonlinear $\overline{\text{PBE}^2}$, but with a surprising choice for the parameterization of h: using the gradient of the value estimate as the features. The nonlinear $\overline{\text{PBE}^2}$ is defined as [Maei et al., 2009]

$$\begin{aligned}&\text{nonlinear } \overline{\text{PBE}^2}(w)\\ &\qquad= \mathbb{E}_\pi\left[\delta(w)\nabla_w\hat{v}(s)\right]^\top\mathbb{E}_\pi\left[\nabla_w\hat{v}(s)\nabla_w\hat{v}(s)^\top\right]^{-1}\mathbb{E}_\pi\left[\delta(w)\nabla_w\hat{v}(s)\right].\end{aligned}$$

This corresponds to the linear $\overline{\text{PBE}^2}$ when $\mathcal{V} = \mathcal{L}$, because $\nabla_w\hat{v}(s) = \mathbf{x}(s)$. Define set $\mathcal{G}_w = \{f : \mathcal{S}\to\mathbb{R} : f(s) = \mathbf{y}(s)^\top\theta, \theta\in\mathbb{R}^d \text{ and } \mathbf{y}(s) = \nabla_w\hat{v}(s)\}$. Notice that this function set for h changes as w changes. Then we get that

$$\theta^*_{\text{nl}} = \arg\min_{h\in\mathcal{G}_w} \mathbb{E}_d\left[(\Delta(s) - h(s))^2\right]$$

satisfies $\theta^*_{\text{nl}}(s) = \nabla_w\hat{v}(s)^\top\theta^*_{\text{nl}}$ where $\theta^*_{\text{nl}} = \mathbb{E}_d\left[\nabla_w\hat{v}(s)\nabla_w\hat{v}(s)^\top\right]^{-1}\mathbb{E}_d\left[\delta(w)\nabla_w\hat{v}(s)\right]$.

Plugging this optimal h back into the formula, we get that

$$\begin{aligned}
&\max_{h \in \mathcal{G}_w} \mathbb{E}_d \left[2\Delta(s)h(s) - h(s)^2 \right] \\
&\quad = \mathbb{E}_d \left[2\Delta(s)\theta^*_{\text{nl}}(s) - \theta^*_{\text{nl}}(s)^2 \right] \\
&\quad = \left(\mathbb{E}_d \left[2\Delta(s)\nabla_w \hat{v}(s)^\top \right] \right) \theta^*_{\text{nl}} - \mathbb{E}_d \left[(\theta^*_{\text{nl}})^\top \nabla_w \hat{v}(s) \nabla_w \hat{v}(s)^\top \theta^*_{\text{nl}} \right] \\
&\quad = 2\mathbb{E}[\delta(w)\nabla_w \hat{v}(s)]^\top \theta^*_{\text{nl}} - (\theta^*_{\text{nl}})^\top \mathbb{E}[\nabla_w \hat{v}(s) \nabla_w \hat{v}(s)^\top] \theta^*_{\text{nl}} \\
&\quad = 2\text{nonlinear } \overline{\text{PBE}^2}(w) - \text{nonlinear } \overline{\text{PBE}^2}(w) \\
&\quad = \text{nonlinear } \overline{\text{PBE}^2}(w)
\end{aligned}$$

This nonlinear $\overline{\text{PBE}^2}$ is not an instance of the generalized $\overline{\text{PBE}^2}$, as we have currently defined it, because the $\mathcal{H}$ changes with w. It is possible that such a generalization is worthwhile, as using the gradient of the values as features is intuitively useful. Further, interchangeability should still hold, as the exchange of the maximum was done for a fixed w. Therefore, it is appropriate to explore an $\mathcal{H}$ that changes with w, and in our experiments we test $\mathcal{H} = \mathcal{G}_w$.

In summary, in this section we introduced the generalized $\overline{\text{PBE}^2}$ and highlighted connections to the linear $\overline{\text{PBE}^2}$ and $\overline{\text{BE}^2}$. The generalized $\overline{\text{PBE}^2}$ provides a clear path to develop value estimation under nonlinear function approximation, providing a strict generalization of the linear $\overline{\text{PBE}^2}$. Two secondary benefits are that the generalized $\overline{\text{PBE}^2}$ provides a clear connection between the $\overline{\text{BE}^2}$ and $\overline{\text{PBE}^2}$, based on a difference in the choice of projection ($\mathcal{H}$), and resolves the identifiability issue in the $\overline{\text{BE}^2}$.

3.3 The quality of the solution

Naturally, identifiability is only a small part of our desired properties. A proxy objective that uniquely identifies one value function is not useful unless that unique value function is useful according to our original objective, the $\overline{\text{VE}^2}$. This criterion parallels the long-standing question about the quality of the solution under the linear $\overline{\text{PBE}^2}$ versus the $\overline{\text{BE}^2}$. In this section, we revisit this question now in context of the generalized $\overline{\text{PBE}^2}$.

Objectives based on Bellman errors perform *backwards bootstrapping*, where the value estimates in a state s are adjusted both toward the value of the next

state and the value of the previous state. In the case of the $\overline{\text{BE}^2}$, backwards bootstrapping can become an issue when two or more states are heavily aliased and these aliased states lead to successor states with highly different values. Because the aliased states look no different to the function approximator, they must be assigned the same estimated value. For each of these aliased states, backwards bootstrapping forces the function approximator to balance between accurately predicting the successor values for all aliased states, as well as adjusting the successor values to be similar to those of the aliased states.

The $\overline{\text{PBE}^2}$, on the other hand, projects the error for the aliased states, ignoring the portion of the Bellman error that forces the function approximator to balance the similarity between the aliased state value and the successor state value. This allows the function approximator the freedom to accurately estimate the values of the successor states without trading off error in states which it cannot distinguish.

To make this concrete, consider the following 4-state MDP from Sutton et al. [2009]. State A_1 and A_2 are aliased under the features for $\mathcal{V}$. For the linear $\overline{\text{PBE}^2}$, $\mathcal{H} = \mathcal{V}$, and so the states are also aliased when approximating h. For the $\overline{\text{BE}^2}$, they are not aliased for h. A_1 transitions to B and terminates with reward 1. A_2 transition to C and terminates with reward 0. The linear $\overline{\text{PBE}^2}$ results in the correct values for B and C—1 and 0 respectively—because it does not suffer from backwards bootstrapping. The $\overline{\text{BE}^2}$, on the other hand, assigns them values $\frac{3}{4}$ and $\frac{1}{4}$, to reduce Bellman errors at A_1 and A_2. A generalized $\overline{\text{PBE}^2}$ with other $\mathcal{H} \neq \mathcal{V}$ would suffer the same issue as the $\overline{\text{BE}^2}$ in this example, unless the projection $\Pi_{\mathcal{H}}$ mapped errors in the aliased states to zero.

On the other hand, the linear $\overline{\text{PBE}^2}$ can find solutions where the Bellman error is very high, even though the projected Bellman error is zero. Consider the plane of value functions that can be represented with a linear function approximator. The Bellman operator can take the values far off of this surface, only to be projected back to this surface through the projection operator. At the fixed-point, this projection brings the value estimate back to the original values and the distance that the value estimate moved on the plane is zero, thus

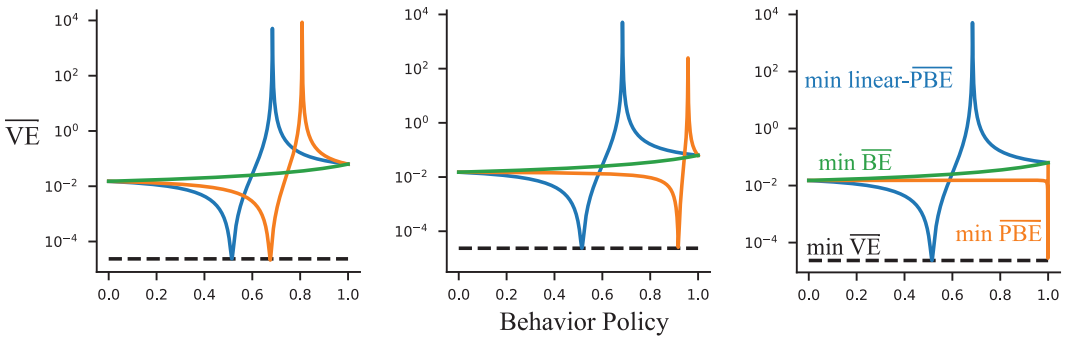

Figure 3.1: The visualization above shows how the $\overline{\text{PBE}^2}$ solution can result in arbitrarily bad $\overline{\text{VE}^2}$ under some behaviors. The vertical axis measures $\overline{\text{VE}^2}$ and the horizontal axis different behavior policies. The blue line above is the same as the visualization used in Kolter [2011] to demonstrate issues with minimizing $\overline{\text{PBE}^2}$. We extend this demonstration show that the $\overline{\text{BE}^2}$ solution (green) exhibits low error and, as the size of $\mathcal{H}_\theta$ grows, the bound on the $\overline{\text{VE}^2}$ improves.

the $\overline{\text{PBE}^2}$ is zero. The $\overline{\text{PBE}^2}$ can be zero even when the $\overline{\text{BE}^2}$ is large. Kolter [2011] provides an example where the solution under the $\overline{\text{PBE}^2}$ can be made arbitrarily far from the true value function. We expand on this example in Figure 3.1, and show that the solution under the linear $\overline{\text{PBE}^2}$ can be arbitrarily poor, even though the features allow for an ϵ accurate value estimate and the solution under the $\overline{\text{BE}^2}$ is very good.

Figure 3.1 illustrates the relationship between the $\overline{\text{VE}^2}$ our various proxy objectives—the linear $\overline{\text{PBE}^2}$, the $\overline{\text{BE}^2}$, and the generalized $\overline{\text{PBE}^2}$ for various $\mathcal{H}_\theta$—on a simple toy problem. This toy problem, proposed by Kolter [2011], provides a clear counterexample for the use of the linear $\overline{\text{PBE}^2}$ as a proxy for the $\overline{\text{VE}^2}$. For a particular combination of behavior policy, target policy, and discount factor, the solution to the linear $\overline{\text{PBE}^2}$ has unbounded error under the $\overline{\text{VE}^2}$. We extend this figure to include the $\overline{\text{BE}^2}$ and the generalized $\overline{\text{PBE}^2}$, where each sub-figure shows the generalized $\overline{\text{PBE}^2}$ with a projection onto a different set, $\mathcal{H}_\theta$, increasing from left to right. When the size of the set $\mathcal{H}_\theta$ is only marginally larger than $\mathcal{V}$, the generalized $\overline{\text{PBE}^2}$ and the linear $\overline{\text{PBE}^2}$ behave similarly. As the size of $\mathcal{H}_\theta$ approaches $\mathcal{F}$, the generalized $\overline{\text{PBE}^2}$ and the $\overline{\text{BE}^2}$ behave similarly, and it is well-established that the $\overline{\text{BE}^2}$ bounds the $\overline{\text{VE}^2}$ [Tsitsiklis and Van Roy, 1997, Maillard et al., 2010].

3.4 The quality of the solution under different weightings

The use of the biconjugate reformulation allows us to easily generalize the function class $\mathcal{H}$ onto which the $\overline{\text{PBE}^2}$ projects. This projection plays the role of emphasizing and de-emphasizing states in the average error [Schoknecht, 2003]. In this section, we consider an alternative generalization of the $\overline{\text{PBE}^2}$ where we directly alter the weighting over states d. In particular, we consider three weightings previously used in the literature: the steady-state distribution given by the target policy d_π, the steady-state distribution given by the behavior d_b, and the effective state weighting produced by emphatic TD $d_{\mathbf{m}}$.

We empirically investigate the quality of the solution under the $\overline{\text{PBE}^2}$ and $\overline{\text{BE}^2}$ with these three different weightings. The solution quality is measured by the $\overline{\text{VE}^2}$ under d_b and d_π. We compute the fixed-point of each objective on a 19-state random walk with randomly chosen target and behavior policies. To isolate the impact of representation on the fixed-points, we investigate several forms of state representation where v_π is outside the representable function class. We include the *Dependent* features from Sutton et al. [2009], randomly initialized sparse ReLU networks, tile-coded features, and state aggregation.

The random-walk has 19 states with the left-most and right-most state being terminal. The reward function is zero everywhere except on transitioning into the right-most terminal state where the agent receives +1 reward, and on the left-most terminal state where the agent receives -1 reward. The discount factor is set to $\gamma = 0.99$.

We run each experimental setting one million times with a different randomly initialized neural network, random offset between tilings in the tile-coder, and randomly sampled target and behavior policy. The policies are chosen uniformly randomly on the standard simplex. The neural network is initialized with a Xavier initialization [Glorot and Bengio, 2010], using 76 nodes in the first hidden layer and 9 nodes in the final feature layer. Then 25% of the neural network weights are randomly set to zero to encourage sparsity between connections and to increase variance between different randomly gen-

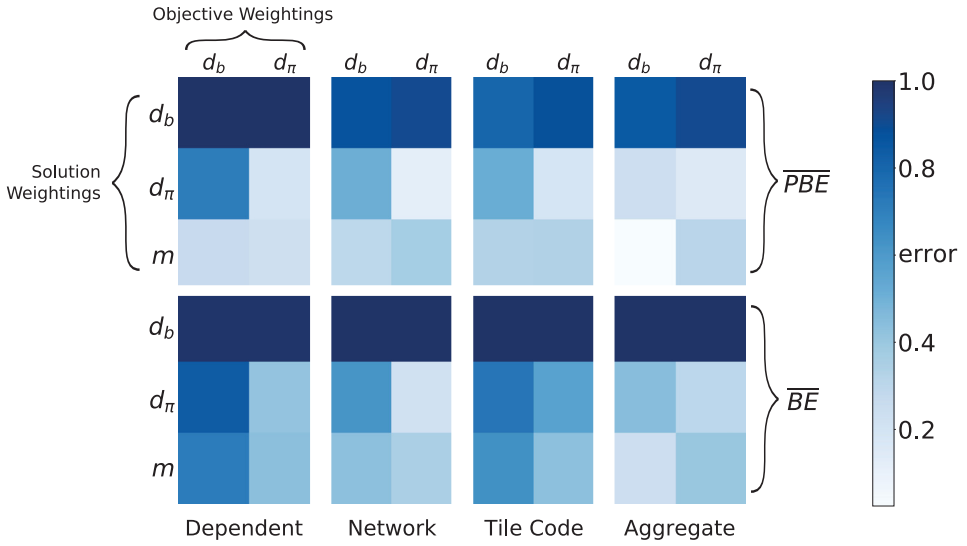

Figure 3.2: Investigating the $\overline{\text{VE}^2}$ of the fixed-points of $\overline{\text{PBE}^2}$ and $\overline{\text{BE}^2}$ under d_b, d_π, and $\mathbf{m}$ on a 19-state random walk. All errors are computed closed form given access to the reward and transition dynamics. The fixed-point of the $\overline{\text{PBE}^2}$ with emphatic weighting consistently has the lowest error across several state representations (light color); while the fixed-point of the $\overline{\text{PBE}^2}$ under d_b has the highest error (dark blue). Results are averaged over one million randomly generated policies and state representations.

erated representations. The tile-coder uses 4 tilings each offset randomly and each containing 4 tiles. The state aggregator aggressively groups the left-most states into one bin and the right-most states into another, creating only two features.

Figure 3.2 shows the normalized log-error of the fixed-points of the $\overline{\text{PBE}^2}$ and $\overline{\text{BE}^2}$ under each weighting. A normalized error between $[0, 1]$ for each representation is obtained by (1) computing the best value function representable by those features $\min_{v \in \mathcal{V}} \overline{\text{VE}^2}(v)$ under d_b or d_π and (2) subtracting this minimal $\overline{\text{VE}^2}$ and normalizing by the maximum $\overline{\text{VE}^2}$ for each column (across objectives and weightings for a fixed representation). The fixed-points are computed using their least-squares closed form solutions given knowledge of the MDP dynamics. Plotted is the mean error across the one million randomly initialized experimental settings. The standard error between settings is negligibly small.

Interestingly, the fixed-points corresponding to weighting d_b consistently

have the highest error across feature representations, even on the excursion $\overline{\text{VE}^2}$ error metric with weighting d_b. The $\overline{\text{PBE}^2}$ under emphatic weighting, $\mathbf{m}$, consistently has the lowest error across all feature representations, though is slightly outperformed by $\overline{\text{PBE}^2}$ with weighting d_π for the $\overline{\text{VE}^2}$ with weighting d_π. In these experiments, the $\overline{\text{BE}^2}$ appears to have no advantages over the $\overline{\text{PBE}^2}$, meaning that the more restricted $\mathcal{H}$ for $\overline{\text{PBE}^2}$ produces sufficiently high quality solutions.

3.5 Summary

This chapter established the framework of biconjugate Bellman objectives that form the basis of our extension into sound value function learning algorithms with nonlinear function approximation. We show a clear relationship between biconjugate Bellman errors and projected Bellman errors, where biconjugate Bellman errors are a strict generalization of the projected variants. This generalization allows us to specify a broad range of objectives with varying properties. We empirically explore the behavior of the fixed-points of the most promising objectives when faced with a limited function approximation class, showing that the relationship between these proxy biconjugate objectives and the true target objective, the $\overline{\text{VE}^2}$, is not always clear.

Similar to the projected Bellman error, the biconjugate Bellman error allows us to avoid issues of double sampling by replacing a sample with an optimization. Because this inner optimization is estimated with limited data and function approximation capacity, we will only ever approximate the projected Bellman error. By contrast, the biconjugate Bellman error allows us to define an objective over both the inner and outer optimizations; a min-max objective over both θ and w. This allows us to characterize the entire space of this saddlepoint optimization, even when using an approximation for the inner optimization.

Finally, we show that the use of the biconjugate error resolves the issue of learnability suffered by the $\overline{\text{BE}^2}$. Because the biconjugate error can be seen as a form of projection, the primary learning process for w is informed

by a secondary process θ that resides in a parametric function class. The $\overline{\text{BE}^2}$, on the other hand, informs the primary learning process w, with an unrealizable process h that depends on privileged state information of the partially observable MDP. The use of biconjugates gives a general tool to construct a wide range of Bellman errors which are learnable. In the next chapter, we will define sample-based algorithms that minimize the biconjugate Bellman error.

Chapter 4

Gradient Temporal Difference with Regularized Corrections

In this section, we describe an algorithm to minimize the biconjugate Bellman error. Because we care about the continual learning setting, we prefer algorithms whose computation and memory are linear in the number of features. We build on the temporal difference (TD) learning algorithm [Sutton, 1988] and the gradient TD family of algorithms [Sutton et al., 2009], which are all online, off-policy learning algorithms.

We start by reviewing the gradient TD family of algorithms, namely TDC and GTD2, and show that these algorithms minimize the biconjugate Bellman error for a particular choice of projection set, $\mathcal{H}$. Then, we demonstrate empirically that TDC has two modes of operation, requiring either extensive tuning or domain knowledge in order to achieve good performance. Using this bimodal perspective of TDC, we derive a new algorithm which minimizes the biconjugate Bellman error which we call the *temporal difference with regularized corrections* algorithm, or TDRC. We investigate the performance of the TDRC algorithm, showing that it performs as well as TD in cases that TD converges, and performs as well as TDC otherwise.

4.1 Gradient TD Methods

Prior methods sought to approximate the gradient of the $\overline{\text{PBE}^2}$ in order to achieve convergence through stochastic gradient descent on a known objec-

tive. There are several ways to approximate and simplify these gradients, each resulting in a different algorithm. The two most well-known approaches are TD with Corrections (TDC) and Gradient TD (GTD2). Both require double the computation and storage of TD, and employ a second set of learned weights $\theta \in \mathbb{R}^d$ with a different stepsize parameter $\eta\alpha_t$, where η is a tunable constant. The updates for the TDC algorithm otherwise are similar to TD:

$$\begin{aligned} w_{t+1} &\leftarrow w_t + \alpha_t \rho_t \delta_t \mathbf{x}_t - \alpha_t \rho_t \gamma (\theta_t^\top \mathbf{x}_t) \mathbf{x}_{t+1} \\ \theta_{t+1} &\leftarrow \theta_t + \eta \alpha_t \big[\rho_t \delta_t - (\theta_t^\top \mathbf{x}_t) \big] \mathbf{x}_t. \end{aligned} \tag{4.1}$$

The GTD2 algorithm uses the same update for θ_t, but the update to the primary weights is different:

$$w_{t+1} \leftarrow \ w_t + \alpha_t \rho_t (x_t - \gamma x_{t+1}) (\theta_t^\top x_t). \tag{4.2}$$

The Gradient TD algorithms are not widely used in practice and are considered difficult to use. Attempts to improve Gradient TD methods have largely come from re-deriving GTD2 using a saddlepoint formulation of the $\overline{\text{PBE}^2}$ [Mahadevan et al., 2014]. This formulation enables us to view GTD2 as a one-time scale algorithm with a single set of weights $[w, \theta]$ using a single global stepsize parameter. In addition, this formulation allows combining GTD2 with acceleration techniques like Mirror Prox [Mahadevan et al., 2014] and stochastic variance reduction methods such as SAGA and SVRG [Du et al., 2017]. Unfortunately, GTD2 endowed with Mirror Prox has never been shown to improve performance over vanilla GTD2 [White and White, 2016, Ghiassian et al., 2018]. Current variance reduction methods like SAGA are only applicable in the offline setting, and extension to the online setting would require new methods [Du et al., 2017].

4.2 Connection to biconjugate errors

To see why (at least) two classes of algorithms arise, consider the gradient for the generalized $\overline{\text{PBE}^2}$, for a given $h(s) \approx \mathbb{E}_\pi \left[\delta(w) \mid S = s \right]$ with a stochastic

sample $\delta(w)$ from $S = s$:

$$-\nabla_w \delta(w) h(s) = h(s)[\nabla_w \hat{v}(s) - \gamma \nabla_w \hat{v}(S')].$$

This is the standard *saddlepoint update*. The key issue with this form is that any inaccuracy in h has a big impact on the update of w, and h can be highly inaccurate during learning. Typically, it is initialized to zero, and so it multiplies the update to the primary weights by a number near zero, making learning slow.

The *gradient correction update* is preferable because it relies less on the accuracy of h. The first term uses only the sampled TD-error.

$$\Delta w \leftarrow \delta(w) \nabla_w \hat{v}(s) - h(s) \gamma \nabla_w \hat{v}(S')$$

where $\Delta\theta \leftarrow (\delta(w) - h(s)) \nabla_\theta h(s)$ just like the saddlepoint update. But, the update is biased because it assumes it has optimal $h^* \in \mathcal{H}$ for part of the gradient. To see why, we extend the derivation for the linear setting.

$$\begin{aligned}
-\nabla_w \delta(w) h(s) &= h(s)[\nabla_w \hat{v}(s) - \gamma \nabla_w \hat{v}(S')] \\
&= h(s) \nabla_w \hat{v}(s) - h(s) \gamma \nabla_w \hat{v}(S') \\
&= (h(s) - \delta(w) + \delta(w)) \nabla_w \hat{v}(s) - h(s) \gamma \nabla_w \hat{v}(S') \\
&= \delta(w) \nabla_w \hat{v}(s) + (h(s) - \delta(w)) \nabla_w \hat{v}(s) - h(s) \gamma \nabla_w \hat{v}(S')
\end{aligned}$$

This resembles the gradient correction update, except it has an extra term $(h(s) - \delta(w)) \nabla_w \hat{v}(s)$. In the linear setting, if we have the linear regression solution for h^* with parameters θ, then this second term is zero in expectation. This is because $\nabla_w \hat{v}(s) = \mathbf{x}(s)$, giving

$$\mathbb{E}_\pi \left[(h(s) - \delta(w)) \nabla_w v(s, w) \mid S = s \right] = \mathbf{x}(s) \mathbf{x}(s)^\top \theta - \mathbf{x}(s) \mathbb{E}_\pi \left[\delta(w) \mid S = s \right]$$

and so in expectation across all states, because $\theta = \mathbb{E}[\mathbf{x}(S)\mathbf{x}(S)^\top]^{-1} \mathbb{E}[\mathbf{x}(S)\delta]$, we get that

$$\begin{aligned}
&\mathbb{E}[(h(S) - \delta(w)) \nabla_w v(S, w)] \\
&\qquad = \mathbb{E}[\mathbf{x}(S)\mathbf{x}(S)^\top] \mathbb{E}[\mathbf{x}(S)\mathbf{x}(S)^\top]^{-1} \mathbb{E}[\mathbf{x}(S)\delta(w)] - \mathbb{E}[\mathbf{x}(S)\delta(w)] \\
&\qquad = \mathbb{E}[\mathbf{x}(S)\delta(w)] - \mathbb{E}[\mathbf{x}(S)\delta(w)] = 0.
\end{aligned}$$

Therefore, given the optimal $h \in \mathcal{H}$ for $\mathcal{H}$ the set of linear functions, this term can be omitted in the stochastic gradient and still be an unbiased estimate of the full gradient.

Unlike the saddlepoint update, however, the gradient correction update is no longer a straightforward gradient update, complicating analysis. It is possible, however, to analyze the dynamical system underlying these updates. The joint update is rewritten as a linear system that is then shown to be a contraction that iterates towards a stable solution [Maei, 2011]. The extension to the nonlinear setting is an important open problem.

4.3 The two modes of TDC

The TDC algorithm is widely considered to have better performance compared to GTD2 [Sutton et al., 2009, White and White, 2016, Ghiassian et al., 2018, 2020]. This improved performance, however, appears to be predominately due to a domain-specific tuning parameter, the secondary stepsize. Through careful, domain dependent, tuning of the secondary stepsize, the TDC algorithm exhibits two modes of operations: TD-mode and GTD-mode. Because TDC—temporal difference learning with corrections—is exactly the TD algorithm with a gradient correction, when the correction factor is zero then TDC becomes exactly TD.

The second mode, GTD-mode, occurs when the secondary stepsize is high. For a sufficiently small stepsize, and enough samples, the secondary estimator h should well-approximate the expected TD error, $h(s) \approx \mathbb{E}_\pi \left[\delta \mid s\right]$. For a large stepsize, say a stepsize near 1, $h(s)$ no longer provides a good estimate of the *average* TD error. Instead, the estimator places greater emphasis on temporally recent TD errors and heavily down-weights prior TD errors. In effect, the biased estimate more closely resembles $h(s) \approx \delta_t(s)$. Making this replacement in the TDC update rule yields the biased residual gradient method, GTD.

The bimodality of TDC's behavior has colored the conclusions drawn from many prior works. Previous empirical investigations have initialized the secondary weights to zero, $\theta_{t=0} = \mathbf{0}$, and tuned the secondary stepsize per prob-

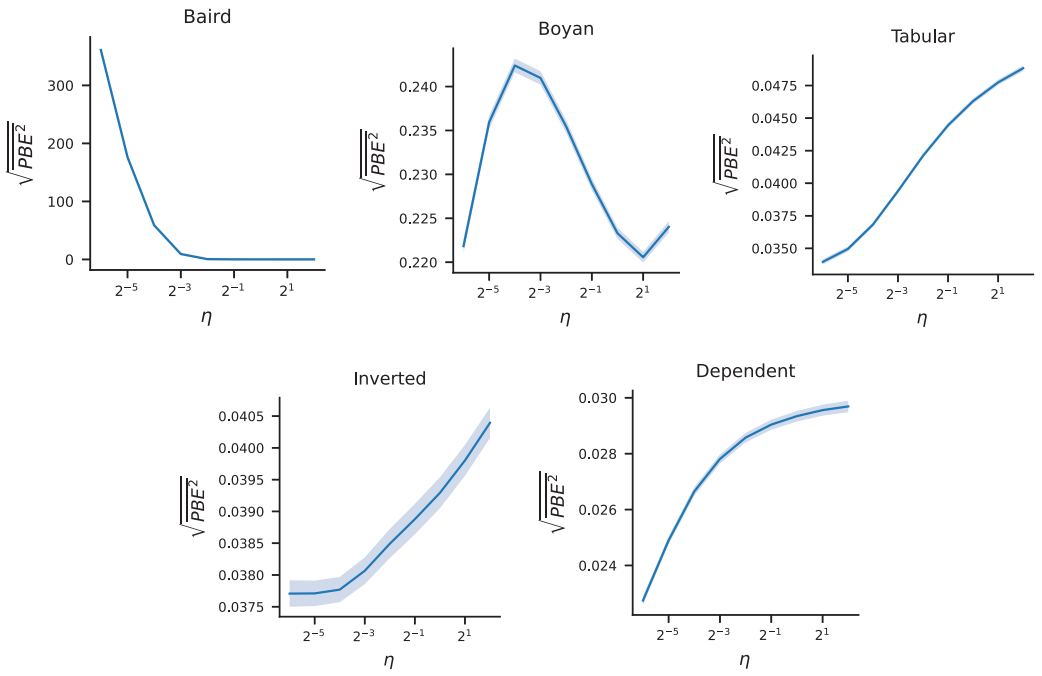

Figure 4.1: The square root of the $\overline{\text{PBE}^2}$ for the TDC algorithm across multiple levels of the secondary stepsize multiplier, η. The solid line denotes the mean performance over 200 independent samples for each level of η and the shaded regions correspond to 95% percentile bootstrap confidence intervals. The shape of the sensitivity curve for the first two domains is opposite that of the final three domains. That is, the strategy for selecting η changes per-domain.

lem. On simple problems, such as random walk domains, a fast and potentially unstable update performs best—such as the TD algorithm or TDC with near-zero correction factor. In these domains, prior works reported TDC with a small secondary stepsize [Sutton et al., 2009, White and White, 2016, Ghiassian et al., 2018]. In more challenging domains, such as Boyan's chain [Boyan, 2002] and Baird's counterexample [Baird, 1999], highly stable and slower updates perform best—such as the GTD2 algorithm or the TDC algorithm with a highly adaptive correction factor. These effects are particularly prevalent when the experiments are run for only a limited number of learning steps.

We can empirically test this hypothesis—that TDC exhibits two behavior modes depending on its secondary stepsize—by investigating TDC's sensitivity as we slowly sweep the secondary stepsize from very small to very large. Most

prior works have used the same set of domains to test gradient TD methods, so we can adopt these same choices for our empirical investigation. We go into more detail about the domains later in Section 4.6

Before we can run this experiment, we first need to deal with the strongly confounding factor of the primary stepsize. We should expect every level of the secondary stepsize to interact with our choice of primary stepsize—choosing a large primary stepsize likely requires choosing a similarly large secondary stepsize, and so on. We deal with this confounding factor in two ways. First, instead of directly sweeping the secondary stepsize, we sweep a multiplicative factor of the primary stepsize, $\beta = \eta\alpha$ for primary stepsize α and secondary stepsize β. Secondly, for every level of η, we tune α independently. In this way, we observe the idealized performance of TDC for each level of η.

Figure 4.1 shows the performance of TDC at multiple levels of the secondary stepsize multiplier, η. We define performance, here, as the square root of TDC's objective function, the $\overline{\text{PBE}^2}$. We average this performance measure over 25k learning steps for each individual agent, then average again over 200 individual agents. The shaded region shows 95% confidence intervals over agents, computed using percentile bootstrap confidence intervals.[1]

In the two challenging domains, Baird's counterexample and Boyan's chain, we see that TDC typically prefers more stable updates with significantly worse performance using small η and better performance with large η. There is an interesting artifact in Boyan's chain where sufficiently tiny η suddenly starts performing well again. We will see in Figure 4.2 that this is an artifact of using such a small number of learning steps. Increasing the length of the experiment gives a sensitivity similar to Baird's counterexample.

The remaining three domains—all forms of an off-policy 5-state random walk—have an opposite sensitivity curve. In these domains, TDC tends to significantly prefer smaller values of η, with a sharp performance drop as η increases.

[1]In this case, the performance metric was approximately normal allowing us to compute less conservative confidence intervals. Because the intervals are already tight, for consistency we will use percentile bootstrap intervals throughout this thesis.

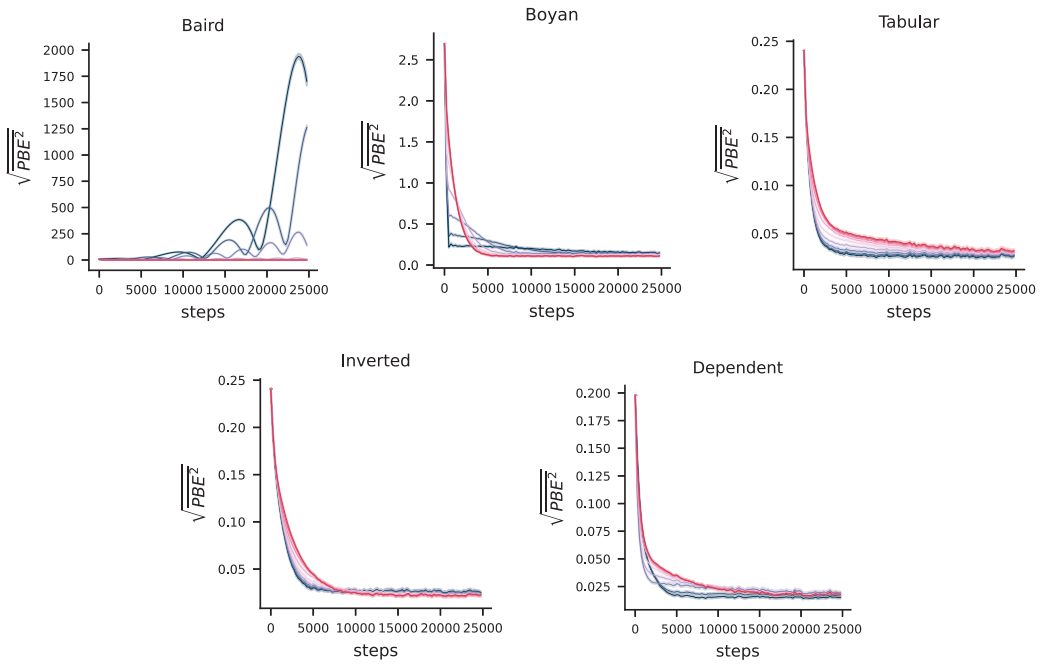

Figure 4.2: The square root of the $\overline{\text{PBE}^2}$ for the TDC algorithm over time and across multiple levels of the secondary stepsize multiplier, η. Darker colors (blue) are small values of η and brighter colors (red) are large values of η. As in Figure 4.1, the first two domains favor agents with a large value of η while the final three domains favor agents with a small value of η.

Figure 4.2 provides a different view of the same data. Here, we expand over the time axis, showing the average performance at every level of η over time. Darker colors (blue) represent small values of η and lighter colors (red) represent large value of η. In Baird's counterexample, we observe large oscillations in performance for small values of η. This is unsurprising, we know the TD algorithm diverges in Baird's counterexample and so should expect TDC in TD-mode to behave similarly. The curves for Boyan's chain illustrate the drawbacks of such short experiments, we selected a large primary stepsize for small values of η resulting in very fast learning with a high plateau. Because the experiment is short, the plateau accounts for only a small amount of the average loss over time. The random walks, where TDC prefers TD-mode, show a smooth degradation of performance as η grows larger. In all three cases, every learning curve converges to the same point given enough data, but the smaller choices for η tend to converge more quickly.

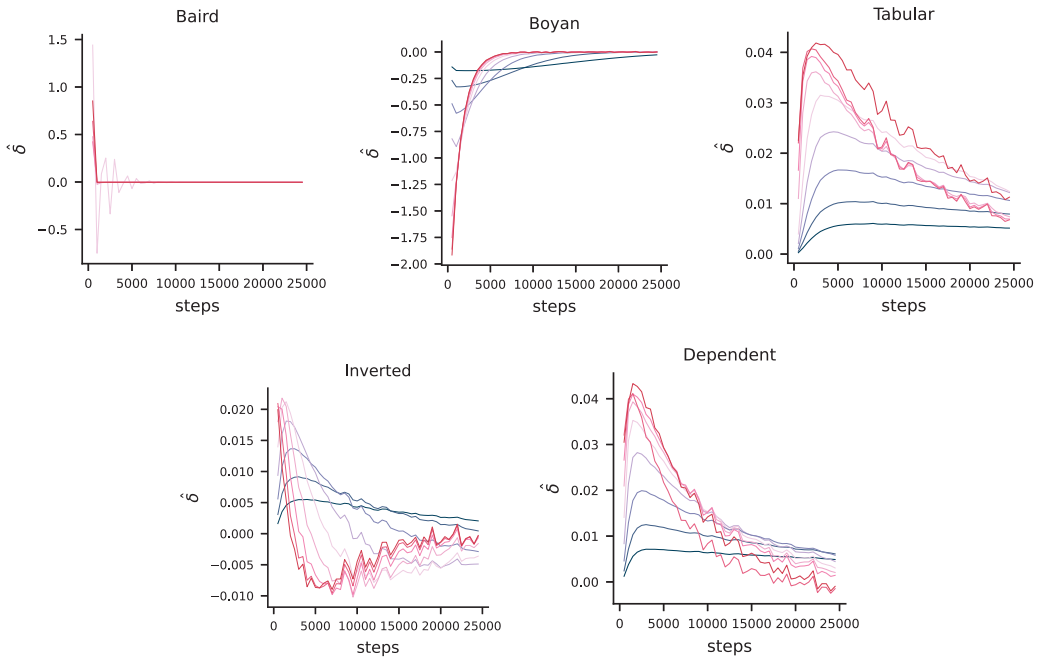

Figure 4.3: Estimated values of the TD error, δ over time for multiple levels of the secondary stepsize multiplier, η. Darker colors are small values of η and brighter colors are large values of η. The primary stepsize was swept independently for every level of η, such that each curve represents near-ideal performance for each given choice of η.

Finally, Figure 4.3 shows the evolution of $h(s) = \hat{\delta} \approx \mathbb{E}_\pi [\delta \mid s]$ for each choice of η. As hypothesized, in all cases the small values of η (blue) stay in a small region around zero, while large values of η (red) tend to stray much further from zero. In the cases that the primary estimator, $\hat{v}$, converges, then all estimates of δ tend towards zero. This highlights that the modality of TDC can shift with time, as the primary estimator converges. Unfortunately, once the primary estimator has reached a point near convergence, the estimate is ambivalent to the presence (or lack thereof) of the gradient correction term—gradients and their corrections are already near zero.

These results imply that TDC is generally a poor learning algorithm. In order to use TDC effectively, we need to know ahead of time which mode to use. However, if we knew this, we would be able to select a better algorithm than TDC. If TDC favors TD-mode for our problem, we would typically be better off using TD; likewise, if TDC favors GTD-mode, we would typically

be better off using GTD2. In this way, we have off-loaded an important part of the learning problem onto a problem of configuration. Instead, we would far prefer an algorithm which uses experiential data to select the appropriate mode adaptively.

4.4 The TDRC algorithm

In this section, we introduce the TDRC algorithm which applies a simple modification to the TDC algorithm in order to adaptively switch between modes of operation using experiential data. We take advantage of the fact that $h(s)$ is estimating the moving target $\mathbb{E}_\pi[\delta_t \mid s]$ for both GTD2 and TDC. As w converges, δ_t approaches zero and consequently θ goes to $\mathbf{0}$ as well. This suggests that an unconstrained linear regression estimate is not necessarily the most efficient choice. In fact, using ridge regression—ℓ_2 regularization—can provide a better bias-variance trade-off: it can significantly reduce variance without incurring any asymptotic bias.

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

The resulting update equations for TDRC are

$$\theta_{t+1} \leftarrow \theta_t + \alpha\big[\rho_t\delta_t - (\theta_t^\top \mathbf{x}_t)\big]\mathbf{x}_t - \alpha\beta\theta_t \tag{4.3}$$

$$w_{t+1} \leftarrow w_t + \alpha\rho_t\delta_t\mathbf{x}_t - \alpha\rho_t\gamma(\theta_t^\top \mathbf{x})\mathbf{x}_{t+1}. \tag{4.4}$$

The update to w is the same as TDC, but the update to θ now has the additional term $\alpha\beta\theta_t$ which corresponds to the gradient of the ℓ_2 regularizer. The updates only have a single shared stepsize, α, rather than a separate stepsize for the secondary weights θ. We find empirically that this choice is

effective, and that the new configuration parameter β sufficiently controls the magnitude of the secondary weights.

While there are many approaches to control the magnitude of the estimator, θ, we use an ℓ_2 regularizer because (1) using the ℓ_2 regularizer ensures the set of solutions for TDRC match TD; (2) the resulting update is asymptotically unbiased, because it biases towards the known asymptotic solution of θ; and (3) the strongly convex ℓ_2 regularizer improves the convergence rate. TDC convergence proofs impose conditions on the size of the stepsize for θ to ensure that it converges more quickly than the "slow-learner" w, and so increasing convergence rate for θ should make it easier to satisfy this condition. Additionally, the ℓ_2 regularizer biases the estimator θ towards $\theta = \mathbf{0}$, the known optimum of the learning system as w converges. This means that the bias imposed on θ disappears asymptotically, changing only the transient trajectory.

4.5 TDRC convergence

In this section, we prove the convergence of the TDRC algorithm using a standard stability argument for stochastic dynamical systems [Borkar and Meyn, 2000]; the same strategy used by other gradient TD methods [Maei, 2011, Sutton et al., 2009]. The argument first handles the stochasticity in the update rule by showing that successive applications of the update are systematically unperturbed by the noise in the system on average. Then the only systematic changes in the dynamical system stem from the expected update, or the *mean-path* update. Because this mean-path update can be shown to be a first-order linear system, we can analyze the spectrum of the system's defining matrix and show that it is a contraction.

Theorem 2 (Convergence of TDRC) *Consider the TDRC update, with a TDC-like stepsize multiplier $\eta \geq 0$:*

$$\theta_{t+1} = \theta_t + \eta\alpha_t \left[\rho_t\delta_t - \theta_t^\top \mathbf{x}_t\right] \mathbf{x}_t - \eta\alpha_t\beta\theta_t, \tag{4.5}$$

$$w_{t+1} = w_t + \alpha_t\rho_t\delta_t\mathbf{x}_t - \alpha_t\rho_t\gamma(\theta_t^\top \mathbf{x}_t)\mathbf{x}_{t+1}, \tag{4.6}$$

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

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

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

In this thesis, we will present a high-level proof sketch and relegate most details to Ghiassian et al. [2020]. We start by combining the TDRC update equations into a single timescale update, $\mathbf{u} \doteq [\theta^\top w^\top]$:

$$\mathbf{u}_{t+1} = \mathbf{u}_t + \alpha_t(\mathbf{G}_{t+1}\mathbf{u}_t + \mathbf{g}_{t+1})$$

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

Assuming the excursion setting, that is all expectations over states are with respect to d_b, then we define $\mathbf{G} = \mathbb{E}_{d_b}[G_t]$ and $\mathbf{g} = \mathbb{E}_{d_b}[g_t]$ and rewrite the above equation as

$$\mathbf{u}_{t+1} = \mathbf{u}_t + \alpha_t\left(\mathbf{G}\mathbf{u}_t + \mathbf{g} + M_{t+1}\right)$$

where M_{t+1} is the stochastic noise sequence. Let the filtration
$\mathcal{F}_t \doteq \sigma(\mathbf{u}_t, M_1, \ldots, \mathbf{u}_{t-1}, M_t)$, then the sequence $(M_t, \mathcal{F}_t)$ is a Martingale difference sequence (MDS) by construction.

In order to use the proof strategy of Borkar and Meyn [2000], it remains to show that (i) the function $h(\mathbf{u}) = \mathbf{G}\mathbf{u}_t + \mathbf{g}$ is Lipschitz in $\mathbf{u}_t$ and there exists the limit $h_\infty(\mathbf{u}) = \lim_{c\to\infty} \frac{h(c\mathbf{u})}{c}$ for all $\mathbf{u} \in \mathbb{R}^{2d}$, (ii) the stepsize sequence α_t satisfies $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 \leq \infty$, (iii) the origin is a globally asymptotically stable equilibrium for the ODE $\dot{\mathbf{u}} = h_\infty(\mathbf{u})$ and (iv) the ODE $\dot{\mathbf{u}} = h(\mathbf{u})$ has a unique globally asymptotically stable equilibrium.

That the function h is Lipschitz is straightforward to show. The condition on stepsizes can be achieved through appropriate construction of stepsize sequences. The challenge remains in showing that conditions (iii) and (iv) hold for mean-path TDRC. The basic strategy is to show that the real parts of the eigenvalues of $\mathbf{G}$ are strictly negative. In the case of TDC (when $\beta = 0$), this is straightforward because $-\mathbf{G}$ is a symmetric matrix. In the case of TDRC (when $\beta > 0$), the regularizer in only the secondary weights causes a spiraling behavior in the iterates, represented by eigenvalues with an imaginary component. Unfortunately, this spiraling behavior requires particular special care and is where a majority of the complexity of the proof lies.

4.6 Experiments in off-policy prediction

We first establish the performance of TDRC across several small linear prediction tasks where we can carefully sweep configuration parameters, analyze sensitivity, and average over many runs. Our goal in this section is to provide evidence for three claims:

1. TDRC has similar performance to TD when TD performs well.
2. TDRC does not diverge in cases when TD diverges.
3. TDRC is insensitive to its configuration parameters compared to TD and GTD methods.

We conduct our investigation in three different toy settings. These problems are designed with varying degrees of state aliasing and target/behavior policy disagreement in order to test these methods under particular component axes. The first problem, Boyan's chain [Boyan, 2002], is a 13 state Markov chain where each state is represented by a compact feature representation. This encoding causes wide generalization during learning, while still allowing v_π to be perfectly represented. Methods with poor bootstrapped estimates will suffer from high transient bias during learning, which we expect to present itself as slow learning.

The second problem is the well-known star counterexample from Baird [1995]. In this MDP, the target and behavior policy are very different resulting in large importance sampling corrections. The extreme aliasing between very different states in the MDP, alongside large importance sampling ratios, cause semi-gradient methods to diverge on this problem. Baird's Counterexample has been used extensively to demonstrate the soundness of Gradient TD algorithms, so provides a useful test bed to demonstrate that TDRC converges empirically.

Finally, we include the five state random walk MDP and feature representations from Sutton et al. [2009]. Two of the three feature representations—tabular and inverted features—allow perfect representation of v_π, with tabular features providing the least generalization across states and inverted features providing the most generalization across states. The third representation—dependent features—cannot perfectly represent v_π, requiring methods to trade off error across heavily aliased states. Like Hackman [2013], we used an off-policy variant of the problem. The behavior policy chooses the left and right action with equal probability and the target policy chooses the right action 60% of the time.

4.6.1 TDRC performs well across problems.

We start by providing empirical evidence in support of the first claim, *TDRC has similar performance TD when TD performs well.* In order to concretely measure this claim, we define "similar performance" as converging to a $\overline{\text{PBE}^2}$ that is within 10% of TD's $\overline{\text{PBE}^2}$ after 3000 online updates. We say that TD "performs well" whenever it converges—meaning across this set of problems we expect TD to only perform poorly on Baird's counterexample domain.

To provide context for TD and TDRC's level of performance, we include several related baseline algorithms—motivating that both TD and TDRC are indeed performing well on these problems. The GTD2 and TDC gradient TD methods [Sutton et al., 2009] are both conceptually similar to TDRC—with TDRC differing from TDC only in its regularization of the secondary weights. Hybrid TD [Hackman, 2013], or HTD, is a related gradient TD method that—

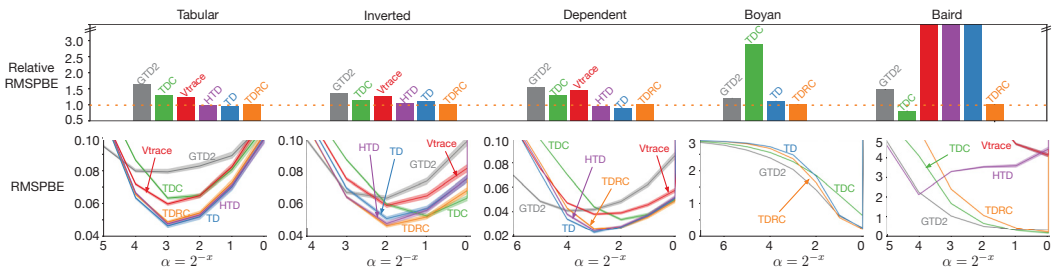

Figure 4.4: **Top:** The normalized average area under the RMSPBE learning curve for each method on each problem. Each bar is normalized by TDRC's performance so that each problem can be shown in the same range. All results are averaged over 200 independent runs with standard error bars shown at the top of each rectangle, though most are vanishingly small. TD and VTrace both diverge on Baird's Counterexample, which is represented by the bars going off the top of the plot. HTD's bar is also off the plot due to its oscillating behavior. **Bottom:** Stepsize sensitivity measured using average area under the RMSPBE learning curve for each method on each problem. HTD and VTrace are not shown in Boyan's Chain because they reduce to TD for on-policy problems.

similar to TDRC—takes advantage of TD's unreasonably good performance across several domains, while incorporating the stability of gradient TD methods. VTrace [Espeholt et al., 2018] is an off-policy learning algorithm built on TD which decreases the variance due to importance sampling updates via clipping.

In order to ensure each method reliably converges within this time-frame, we use the Adagrad optimizer [Duchi et al., 2011] which accelerates early learning, while causing stepsizes to monotonically decrease over time. We first consider idealized performance for each method, using an extensive grid-search over the stepsize parameter for TD, TDRC, VTrace, and HTD; as well as a grid-search over both the primary and secondary stepsizes for GTD2 and TDC. All other configuration parameters—such as the regularization parameter for TDRC—are kept to their default values.

Figure 4.4 shows the $\overline{\text{PBE}^2}$ for each method relative to TDRC; a relative $\overline{\text{PBE}^2}$ near one means a method performed similarly to TDRC and a relative $\overline{\text{PBE}^2}$ near two means that method performed roughly two times worse. On the first four problem settings—those where TD performs well—we see that TDRC consistently performs near-identically to TD, providing strong evidence

to support our claim. On the fifth problem, Baird's counterexample, we see TD's performance is far off the chart as expected. The gradient TD methods typically performed well on Baird's counterexample with TDRC learning slightly slower than TDC.

The ordering over methods is generally as we would expect. GTD2 tended to learn slowly in each domain, leading to a worse final performance in our relatively short timescale of 3000 online updates. Because GTD2 directly minimizes the biconjugate objective, it tends to make highly conservative updates—only modifying its $\hat{v}$ estimate once it has an accurate internal estimate of the TD error. All other investigated methods depend on direct samples of the TD error and so tend to be much less conservative, each at different levels. TD, VTrace, and HTD are insufficiently conservative and can still diverge in settings such as Baird's counterexample, while TDC tends to be more conservative than TDRC and so learns more slowly on problems with friendly conditions. TDRC tends to provide a sensible balance between conservative and anti-conservative updates.

4.6.2 TDRC is insensitive to its configuration parameters.

Having shown that TDRC performs similarly to TD when TD performs well, we now focus on the third primary claim in the online, off-policy prediction setting: "*TDRC is insensitive to its configuration parameters.*" In the previous section, we illustrated the idealized performance of TDRC alongside several baselines, motivating that TDRC *can* perform well when well-configured. This claim, however, is uninteresting in isolation if we have succeeded only in shifting from a learning problem to a configuration problem.

In this section, we investigate the primary configuration parameters of the TDRC algorithm across all five problem settings. We show that TDRC is insensitive to its primary stepsize in comparison to any of the previously investigated baseline algorithms. We show that TDRC smoothly interpolates between TD and TDC depending on its regularization parameter, with intermediate values of the regularization parameter frequently out-performing

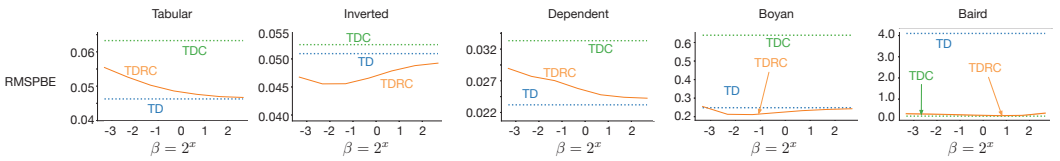

Figure 4.5: Sensitivity to the regularization parameter, β. TD and TDC are shown as dotted baselines, demonstrating extreme values of β; $\beta = 0$ represented by TDC and $\beta \to \infty$ represented by TD. This experiment demonstrates TDRC's notable insensitivity to β. Its similar range of values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

both TD and TDC. And finally, we use a modified version of the random walk problems to highlight that TDRC only exhibits sensitivity to its regularization parameter in extreme instances of this MDP.

In Figure 4.5 we investigate performance across a range of $\beta \in 0.1 * \{2^0, 2^1, \ldots, 2^5, 2^6\}$. We also include two extreme values of β, TDC ($\beta = 0$) and TD (large β). Ideally, performance should quickly improve for any non-negligible β with a large flat region of good performance in the parameter sensitivity plots for a wide range of β. Intuitively, TDRC should not be sensitive to β as both extremes—TDC and TD—generally perform well on most problems. Picking a $\beta > 0$ should generally enable TDRC to learn faster—like TD—by providing a lower variance correction. However, a default choice of β cannot be too large in order to ensure we avoid the divergence issues of TD.

Empirically, we observe in Figure 4.5 that even small values of β—for instance $\beta = 0.1$—tend to significantly outperform TDC across problems. In two cases, TDRC splits the difference between TDC and TD (Random Walk with Tabular or Dependent features) and in three cases, TDRC outperforms both TD and TDC (Random Walk with Inverted Features, Boyan's chain and Baird's counterexample). Notably, the settings where TDRC matches or outperforms TD and TDC are those with more complex feature representations, suggesting that the regularization parameter helps TDRC learn an $\mathbf{h}$ that is less affected by harmful aliasing in the feature representation. Finally, Figure 4.5 also suggests that $\beta = 1.0$ was in fact not optimal, and we could have obtained even better results in the previous section, typically with a larger

β. These improvements, though, were relatively marginal over the choice of $\beta = 1.0$.

Naturally, the scale of β should be dependent on the magnitude of the rewards because the gradient correction term is tracking the expected TD error. To understand the impact of reward magnitude on performance across multiple values of β, we test TDRC on a set of modified random walk environments. We design a set of small experiments to understand how changes in the environment cause the scale of $\mathbf{h}$ to change, and how that relates to the performance of TDRC across several values of β. The scale of $\mathbf{h}$ changes whenever the size of the TD error or scale of the features change.

For these experiments, we increase the range of the TD error by making the initial value function $\hat{v}_{t=0} = \mathbf{0}$ and manipulating the magnitude of the rewards. We run this experiment on the five state random-walk domain with each of the feature representations, and change only the rewards in the terminal states by a multiplicative constant. We compute the mean and standard deviation of TD's performance across 500 independent runs and compute the number of standard deviations TDRC's mean performance is from TD's mean performance. Finally, we let the reward vary by order of magnitudes, with the multiplicative constant taking values $\{10^{-2}, 10^{-1}, \ldots, 10^{3}\}$. For each scaling, we test multiple values of $\beta \in \{2^{-5}, 2^{-4}, \ldots, 2^{4}\}$ and for each of these instances we select the best constant stepsize from $\{2^{-5}, 2^{-4}, \ldots, 2^{-1}\}$.

In Figure 4.6, we show the range of β for which TDRC's performance is as good, or nearly as good, as TD's performance as the magnitude of the rewards increases. As hypothesized, the range of acceptable β decreases as the reward magnitude increases; however, the range of β only appreciably shrinks for a pathologically large deviation between rewards and initial value function. This demonstrates that, while β is problem dependent, its range of acceptable values is robust to all but the most pathological of examples across several representations. One strategy to avoid the need to tune β is to employ an adaptive target normalization, such as Pop-Art [van Hasselt et al., 2016], in order to control the magnitude of rewards and keep β equal to one.

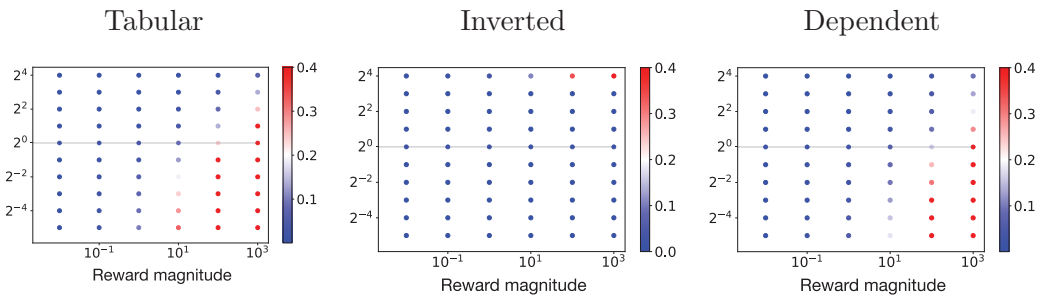

Figure 4.6: Relationship between TDRC and TD performance across different reward scales for different values of beta. On the x-axis we show the scale of the rewards for the terminal states of the random walk, on the y-axis we show a range of values of β. Each dot represents the number of standard deviations away from TD that TDRC's performance is across 500 independent runs for that particular value of β. For each dot, TDRC and TD choose the stepsize with the lowest area under the RMSPBE learning curve; with stepsizes swept from $\alpha \in \{2^{-5}, 2^{-4}, \ldots, 2^{0}\}$. As the scale of the rewards increases (left to right on the x-axis), the variance of the secondary weights, $\mathbf{h}$, also increases; effectively requiring a larger value of β. This figure demonstrates that TDRC with $\beta = 1$ remains relatively insensitive to the scale of the rewards except in extreme cases when the variance of the rewards from transition to transition is quite large.

4.7 Summary

In this chapter, we introduced the temporal difference learning with regularized corrections algorithm (TDRC), which is a simple modification of TDC from Sutton et al. [2009]. The construction of this algorithm came from two predominant insights. First, from Chapter 3, the use of the biconjugate Bellman error establishes that the internal secondary parameters used with gradient TD methods should track the expected TD error, conditioned on features. We can constrain this internal process to a smaller function class, which has the effect of "projecting away" a larger share of the TD error transiently. The second insight, established in this chapter in Section 4.3, is that the TDC algorithm prefers to be configured like TD most of the time, but sometimes prefers to be configured like a gradient TD method. Combining these two insights led to the use of a ridge regression estimator for the secondary set of parameters of the TDC algorithm, using the more aggressive projection to improve stability of the algorithm. This modification takes advantage of the fact that the bias

due to regularization only impacts the transient behavior of the algorithm, but not the asymptotic solution.

We empirically validated this algorithm by first testing it in the suite of diagnostic MDPs previously used to evaluate gradient TD methods. These simple toy problems tested the performance of TDRC under varying degrees of state aliasing or limited function approximation capacity, varying degrees of off-policy data, and on a particular combination of these that provably leads to the divergence of semi-gradient TD methods. We consistently found that TDRC performed well across each of these conditions, typically performing as well as the best baseline or within a narrow margin of the best baseline in every case. By contrast, no other algorithm exhibited consistent performance across this entire test bed. TD performed exceptionally in all cases but Baird's counterexample, while gradient TD methods performed poorly in all cases except Baird's counterexample.

In the next chapter, we will extend TDC to the control problem and introduce the use of neural network function approximation for gradient TD methods.

Chapter 5

Q-Learning with Regularized Corrections

In this chapter, we extend the TDRC algorithm to both nonlinear function approximation and control. We call this algorithm Q-learning with regularized corrections (QRC) because, like TDRC, we use a gradient correction update that is estimated with a nonlinear regularized regression. Finally, we propose a complete learning system by embedding the QRC update rule into the DQN learning system [Mnih et al., 2013], replacing the original Q-learning update and removing target networks.

We empirically investigate the performance of the QRC algorithm, showing that it performs well across three classic control environments as well as a more complex simulation environment. Our results provide strong evidence that the QRC algorithm is empirically more reliable than semi-gradient alternatives, such as DQN, even with the exclusion of target networks. This chapter uses a nonstandard evaluation of performance, comparing the distribution of performance over multiple individual agents to baselines, showing the reliability of gradient TD methods across many ablations. Finally, we introduce a new analytical strategy for evaluation reinforcement learning algorithms, focusing on agent-centric performance and providing a novel view of reliability.

5.1 Extending to non-linear control

We start with the extension of the TDRC algorithm to neural network function approximation. Recall with linear function approximation, the expected TD error is estimated using linear regression with ℓ_2 regularization: $\theta^\top \mathbf{x}_t \approx \mathbb{E}_\pi [\delta_t \mid S_t = s]$. With neural networks, we estimate this expected TD error using an additional head on the network updated with a regularized mean-squared objective and tracking δ_t as its target.

Because this internal process is secondary to value function learning, we want to avoid degrading the performance of the value function estimates simply to improve estimates of $\mathbb{E}_\pi [\delta_t \mid S_t = s]$. To accomplish this, we prevent gradients from the secondary head—the head which tracks δ_t—from passing gradients of its squared error back to the feature-learning layers of the neural network. This choice is made for simplicity, and to avoid any issues when balancing between two losses of differing magnitudes. This choice also makes the connection to TD more clear as β becomes larger, as the update to the network is only impacted by w.

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

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

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

The remaining weights w_t, which include all the weights in the network excluding θ, are updated using

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma q(S_{t+1}, a') - q(S_t, A_t) \\ w_{t+1} &\leftarrow w_t + \alpha\delta_t \nabla_w \hat{q}(S_t, A_t) - \alpha\gamma\hat{\delta}(S_t, A_t)\nabla_w \hat{q}(S_{t+1}, a') \end{aligned} \tag{5.2}$$

where a' is the action that the policy we are evaluating would take in state S_{t+1}. For control, we often select the greedy policy, and so $a' = \arg\max_a q(S_{t+1}, a)$

and $\delta_t = R_{t+1} + \gamma \max_a q(S_{t+1}, a) - q(S_t, A_t)$ as in Q-learning. This action a' may differ from the (exploratory) action A_{t+1} that is actually executed, and so this estimation is off-policy. There are no importance sampling ratios because we are estimating action-values and all updates are conditioned on the action.

5.2 Experiments in non-linear control

In this section, we empirically evaluate the performance of the QRC algorithm on several simulation environments. Having established TDRC's high performance in the linear function approximation setting in Section 4.6, we jump directly to nonlinear function approximation in this section. We compare QRC's performance to two other gradient-based nonlinear control algorithms, Greedy QC with locally linear projections [Maei et al., 2009] and SBEED [Dai et al., 2018]. We additionally include semi-gradient Q-learning as a baseline. In these experiments, we provide evidence for the following claims:

1. QRC performs well across all tested domains when evaluated with idealized performance with respect to its configuration parameters.

2. QRC is insensitive to its configuration parameters, allowing consistent performance without domain-specific tuning.

3. QRC is stable across a wide range of stepsizes.

We chose simulation domains with a sufficiently small state dimension to efficiently compare algorithms across many random neural network initializations, while still providing statistically sound evidence. Despite their small state dimension, we require domains with sufficiently complex learning dynamics in order to tease out differences between each algorithm and parameter configuration under investigation. We chose three classic control domains known to be challenging when approximation resources and agent-environment interactions are limited: Acrobot [Sutton, 1996], Cartpole [Barto et al., 1983], and Mountain Car [Moore, 1990]. We also used Lunar Lander [Brockman et al., 2016] to investigate performance in a domain with a dense reward function and moderately higher-dimensional state.

The network architectures were as follows. For Acrobot and Mountain Car, we used two layer fully-connected neural networks with 32 units in each layer and a ReLU transfer. The output layer has an output for each action-value and uses a linear transfer. For the Cartpole and Lunar Lander domains, we used the same architecture except with 64 units in each hidden layer. We use a shared network with multiple heads for all algorithms unless otherwise specified. In experiments with policy-gradient-based methods the parameterized policy uses an independent neural network. We do not use target networks.

We swept consistent values of the hyperparameters for every experiment across all algorithms. Specifically, we swept the stepsize parameter over a wide range $\alpha \in \{2^{-12}, 2^{-11}, \ldots, 2^{-7}\}$ for every algorithm. For algorithms which chose a stepsize on the boundary of this range—for instance, GQ often chose the smallest stepsize—we performed a one-off test to ensure that the range was still representative of the algorithm's performance. All algorithms used mellowmax, with τ swept in the range $\tau \in \{0, 10^{-4}, 10^{-3}, \ldots, 10^{0}\}$, including 0 to allow algorithms to choose to use a hard-max. Algorithms based on the SBEED update have an additional hyperparameter η which interpolates between the gradient correction update and a residual gradient update. For all experiments we swept values of $\eta \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$ and the ratio between the actor and critic stepsizes $\nu \in \{2^{-4}, 2^{-3}, \ldots, 2^{1}\}$, often giving SBEED algorithms twenty-four times as many parameter permutations to optimize over compared to other algorithms. Likewise, we allowed saddlepoint methods (GQ) to optimize over the regularization parameter $\beta \in \{0, 0.5, 1, 1.5\}$, to give them an opportunity to perform well.

The remaining hyperparameters were not swept, but instead set to reasonable defaults. We used a replay buffer to store the last 4000 transitions, then sampled 32 independent transitions without replacement to compute mini-batch averaged updates. Every algorithm used the ADAM optimizer [Kingma and Ba, 2015] for all experiments with the default hyperparameters, a momentum term of $\beta_1 = 0.9$ and a squared-gradient term of $\beta_2 = 0.999$. We additionally tested Stochastic Gradient Descent and RMSProp and found that most conclusions remain the same, so choose not to include these results to focus

the presentation of results. For each of the four domains we use a discount factor of $\gamma = 0.99$ and cutoff long-running episodes at 500 steps for Acrobot and Cartpole and 1000 steps for Mountain Car. On episode cutoff events, we do not make an update to the algorithm weights to avoid bootstrapping over this imaginary transition and on true episode termination steps we update with $\gamma = 0$.

We use a non-conventional performance measure to more fairly report algorithm performance. A common performance metric is to report the cumulative reward at the end of each episode, running each algorithm for a consistent number of episodes. This choice causes algorithms to have different amounts of experience and updates. Some algorithms use more learning steps in the first several episodes and achieve higher asymptotic performance because they effectively learned for more steps. We instead report the cumulative reward from the current episode on each step of the current episode. For example in Mountain Car, if the *kth* episode takes 120 steps, then we would record -120 for each step of the episode. We then run each algorithm for a fixed number of steps instead of a fixed number of episodes, so that each algorithm gets the same number of learning steps and a consistent amount of data from the environment. We record performance over 100,000 steps, recorded every 500 steps—rather than every step—to reduce storage costs.

To avoid tuning the hyperparameters for each algorithm for every problem, we start by investigating a single set of hyperparameters for each algorithm across all four benchmark domains. We evaluate the hyperparameters according to mean performance over runs, for each domain. We then use a Condorcet voting procedure to find the single hyperparameter setting that performs best across all domains.

5.2.1 QRC performs consistently well across environments.

In this section, we provide evidence that QRC performs consistently well across environments, generally outperforming other related gradient-based approaches and significantly outperforming a typical semi-gradient baseline. We

compare each algorithm under two idealized conditions. The first highlights the performance of every algorithm when configuration parameters can be extensively tuned across all potential domains of interest—for the purposes of this experiment, these are the four simulation domains. This simulates the setting where we have partial information about the application domains where our algorithm may be deployed, but we do not know exactly the properties of the domain at configuration time.

The second setting illustrates the idealized performance of each algorithm when configuration parameters can be extensively tuned for each domain specifically. This simulates the setting where we know exactly the domain where an algorithm will be deployed ahead-of-time, and have a sufficiently large interaction budget in order to perform tuning runs of the algorithm on that domain. Both settings are *idealized*; in most deployment scenarios, one would not have sufficient interaction budget to extensive tune their algorithm to the exact domain where it will be deployed. Because it is inordinately challenging to provide unbiased estimates of idealized performance, we ensure QRC receives a much smaller tuning budget than the other algorithms—if our hypothesis that QRC is less sensitive is true, this should only negligibly harm QRC's performance.

The top subfigure in Figure 5.1 shows the learning curves for each algorithm with the single best performing hyperparameter setting *across* domains. To select this singular hyperparameter configuration, we use a Condorcet voting procedure (particularly Black's ranked-choice vote) with each domain producing a ranking of each hyperparameter configuration for each seed. Hyperparameters were ranked according the area under the learning curve for a given seed. By allowing each domain to cast multiple ranked-choice votes—i.e. one vote per seed—this procedure inherently captures uncertainty due to variation in the ordering over hyperparameter configurations. In fact, a closely related procedure has been linked to performing a ranked hypothesis test [].

QRC was the only algorithm to consistently be among the best performing algorithms on every domain and was the only algorithm with a single configuration that could jointly solve all four domains. Although SBEED was given

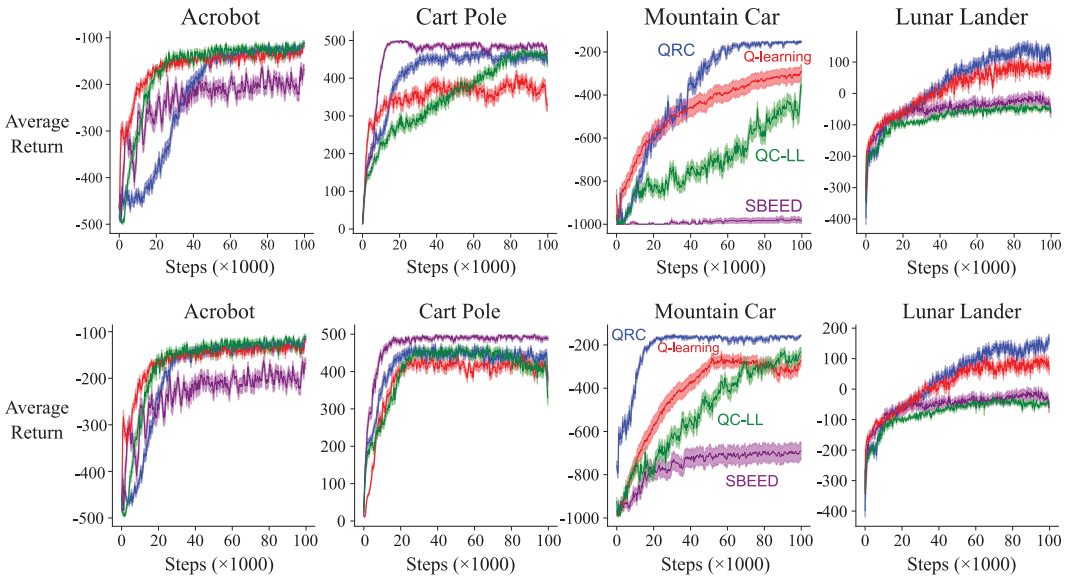

Figure 5.1: **Top:** Best hyperparameters *across* domains. **Bottom:** Best hyperparameters *per*-domain. The performance of several off-policy control algorithms with neural network function approximation on four simulation domains. The top subplot uses a voting procedure to select a single hyperparameter setting for each algorithm, used across all four domains. This gives a sense of idealized performance of each algorithm without domain-specific tuning. The bottom subplot tunes hyperparameters per-domain, giving a sense of idealized performance when each algorithm can tune hyperparameters for each specific domain. The performance of each algorithm in the bottom subplot should be greater or equal to the corresponding performance in the top subplot on average over timesteps. In every domain, at least one algorithm reaches a performant and stable final policy. QRC is among the top-performing algorithms in *every* domain, where all other algorithms have at least one domain where they perform notable worse than the rest. In Cartpole, the three ϵ-greedy based methods (QRC, QC-LL, and Q-learning) suffer from a fixed-entropy behavior policy which causes performance to plateau at a slightly worse average value than the policy gradient method SBEED whose policy becomes near deterministic by the end of learning.

twenty-four times as many hyperparameter combinations to optimize over, its performance was consistently worse than all other benchmark algorithms. This suggests that the voting procedure was unable to identify a single hyperparameter setting that was consistently good across domains. QC-LL performed well on the two simpler domains, Acrobot and Cartpole, but exhibited poor performance in the two more challenging domains. We note, also, that our QC-LL implementation had a markedly slower runtime than other algorithms, requiring approximately double the wall clock time to complete experiments.

The bottom subfigure of Figure 5.1 shows the learning curves for per-domain idealized performance of each algorithm. We again tune over the same set of hyperparameter configurations, however in this plot we select the best hyperparameter configuration independently for each domain. Each hyperparameter configuration is ranked using the average over seeds of the area under the learning curve using 100 seeds for every configuration, domain, and algorithm tuple. We find that QRC generally outperforms all other methods except in the case of Cartpole, where the policy gradient method SBEED plateaus at a slightly higher performance than the value-based methods. This is because SBEED's policy becomes near deterministic, while the value-based methods use a fixed ϵ-greedy behavior policy which can be harmful in the Cartpole domain.

There are many challenges associated with reporting idealized performance of an algorithm that has many configuration parameters [Jordan et al., 2020, Patterson et al., 2023]. Figure 5.1 tells us how each algorithm *can* perform given a fairly large budget to tune configuration parameters. By contrast, Figure 5.2 shows how well each algorithm does over its entire space of configuration parameters. The distributions over performance show how the performance differs when we are uncertain about the optimal configuration of each algorithm. An algorithm which is more sensitive to configuration would have unfavorable degradation when we have not identified the optimal configuration, while less sensitive algorithms will degrade gracefully and will have a large proportion of runs centered around near-optimal performance.

For computational budget reasons, we reuse the same data as for Figure 5.1

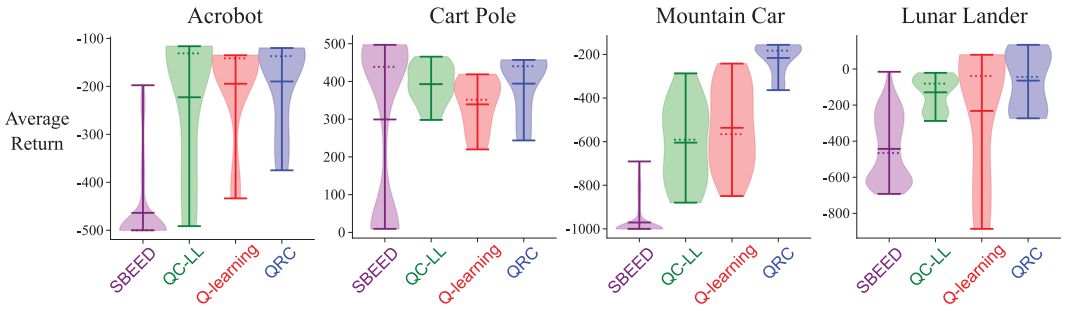

Figure 5.2: *Distribution of average returns over hyperparameter settings for each benchmark domain.* The vertical axis represents the average performance of each hyperparameter setting (higher is better) and the width of each curve represents the proportion of hyperparameters which achieve that performance level, using a fitted kernel density estimator. The solid horizontal bars show the maximum, mean, and minimum performance respectively and the dashed horizontal bar represents the median performance over hyperparameters. QRC in blue generally performs best and exhibits less variability across hyperparameter settings.

to plot the performance distributions in Figure 5.2. We can view the grid-search data as a form of bucket sampling from a quota; we define each bucket as a unique cross-product between configuration settings for each parameter, and we collect a strict quota of 100 samples per bucket. As a consequence of discretization, we lose the ability to capture potential performance variability in between measurements. We implicitly assume, then, that the performance of each algorithm is relatively smooth and that our discretization approach has sufficiently high-fidelity to capture all regions of interest. A positive consequence of bucket sampling is that it allows the use of a repeated measures empirical design, allowing us to ignore unrelated variation such as variance in performance due to the environment, stochastic policies, or sampling from a replay buffer. This allows us to focus our attention to the variability due exclusively to configuration.

In Figure 5.2, we observe that QRC is typically far less sensitive to its configuration parameters than the baseline algorithms. Generally, QRC has a large proportion of the configurations which achieve near-best performance, suggesting that minor suboptimality in configuration has negligible impact on performance. By contrast, SBEED is typically highly sensitive to configura-

tion with only a small proportion of configurations achieving a reasonable level of performance and the bulk of configuration setting achieving a performance level no better than a random uniform policy. Neither QC-LL nor Q-learning have a clearly identifiable pattern across domains, suggesting that their configuration parameters—and even their sensitivity—are highly domain-dependent.

5.2.2 QRC is more reliable than DQN.

One of the central claims of this thesis is that gradient-based methods are more reliable than their semi-gradient counterparts. In this section, we provide both direct and indirect evidence to support this claim in the nonlinear, control setting. We start by investigating an indirect resultant of stability: the variability of QRC's performance over experimental trials. We expect a stable algorithm to have low variability across trials. Then, we investigate internal processes within our QRC agents, providing a direct measurement of stability and showing that QRC is indeed more reliable than its semi-gradient counterpart, DQN.

As in the previous sections, we start with the idealized case where we have extensively tuned configuration parameters for each algorithm. In Figure 5.3, we show the performance distribution over experimental trials—also called "agents", "seeds", or "independent runs" in the literature—where the x-axis shows various levels of performance (right-side is better performance) and the y-axis shows the proportion of experimental trials which attained that level of performance. In particular, we show the performance distribution for the Lunar Lander environment, picked because it provides a clear demonstration of this particular form of analysis. The bars in Figure 5.3 show the binned, observed performance values, while the solid curves show a gaussian kernel density estimator fit to the raw observations.

By analyzing the shape of the distributions in Figure 5.3, we can begin to understand the stability properties of each investigated algorithm. We observe that—on this domain—both QRC and Q-learning have fairly consistent performance, centered around a near-optimal policy (Lunar Lander is considered "solved" when an agent achieves a return of 100). A very small share of QRC

agents failed to solve this domain within the given time constrains, as did a slightly larger share of Q-learning agents. By contrast, very few QC-LL agents achieved a satisfactory level of performance with nearly all agents performing suboptimally by the end of the experiment. Importantly, QC-LL agents consistently learned to stop crashing the lander module—a situation which leads to large negative rewards—but typically burned a lot of fuel in order to do so. Finally, we observe that no SBEED agent managed to solve this domain with a large share of agents settling on a very low-performing policy and a small share of agents finding catastrophic policies that consistently crash the lander module.

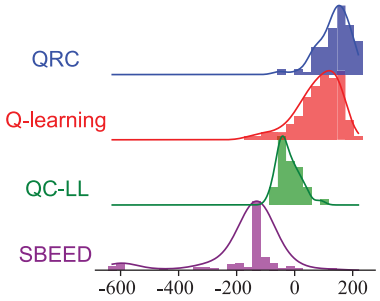

Figure 5.3: *The performance distribution over runs* for the best performing hyperparameter settings for each algorithm on Lunar Lander. The horizontal axis represents the average episodic return over the last 25% of steps. The vertical axis for each subplot represents the proportion of trials that obtained a given level of performance. The plot shows the empirical histogram and kernel density estimator for the performance distribution over 100 independent trials. Mass concentrated to the right indicates better performance.

This resultant effect of stability—the performance variability—is neatly illustrated for idealized configuration parameters and on the singular simulation environment, Lunar Lander. A natural next step is to expand the scope of our analysis. We do this two-fold, by investigating performance variability across each of our four simulation environments and by investigating variability across multiple levels of the stepsize parameter. We choose to expand over stepsizes because the choice in stepsize has the largest impact in performance. It is also natural to conjecture that stepsizes have a strong interaction with stability; one would expect a large stepsize to lead to chasing noise and inhibiting convergence, while an overly small stepsize would have consistent, but

poor, performance.

In Figure 5.4 we plot the performance distribution for every algorithm, for every level of the stepsize parameter, and for every environment. Naturally, this leads to many plots in a small space, but importantly each of the 24 subplots follows the exact same structure as Figure 5.3. The bolded distributions are those for the best stepsize, while the faded distributions show the performance variability for stepsizes off of the ideal configuration; that is, every environment has exactly four bold distributions, one for each algorithm.

We notice a key trend in Figure 5.4, both QRC and QC-LL tend to have consistent and stable performance for appropriately chosen stepsize—particularly for sufficiently small stepsize. While, QC-LL tends to be stable, the number of stepsizes that lead toward stability appears quite narrow, where QRC is stable for every stepsize below a given domain-dependent threshold. For every environment, there is a choice of stepsize for which Q-learning appears stable and reasonable high-performance, suggesting that given sufficient tuning resources Q-learning can be made to work well on these environments. However, for most of these environments, stepsizes that are off from optimal—even by a single level—often exhibit a substantial degradation in performance, stability, or both. Notice, for example, Q-learning on the Mountain Car environment. Every stepsize apart from the optimal stepsize has highly bimodal performance, with some runs failing to outperform a uniform random baseline.

There is an unfortunate limitation with the investigation in Figure 5.4. Because we measure an emergent property of stability (or instability), we suffer from some potential confounding factors. For example, look at the performance of the QC-LL algorithm for any given environment; particularly Mountain Car. As the stepsizes grow larger (towards the bottom of the plot) we notice the variability in performance grows. This naturally seems to imply that QC-LL becomes unstable for these stepsizes—and, in fact, this is true! However, as the stepsizes grow smaller (towards the top of the plot), we notice again that the variability in performance grows. If we again take this to imply that QC-LL is suffering from instability, in this case we would be wrong. Here, QC-LL is instead suffering due to issues of sample efficiency and the fact that

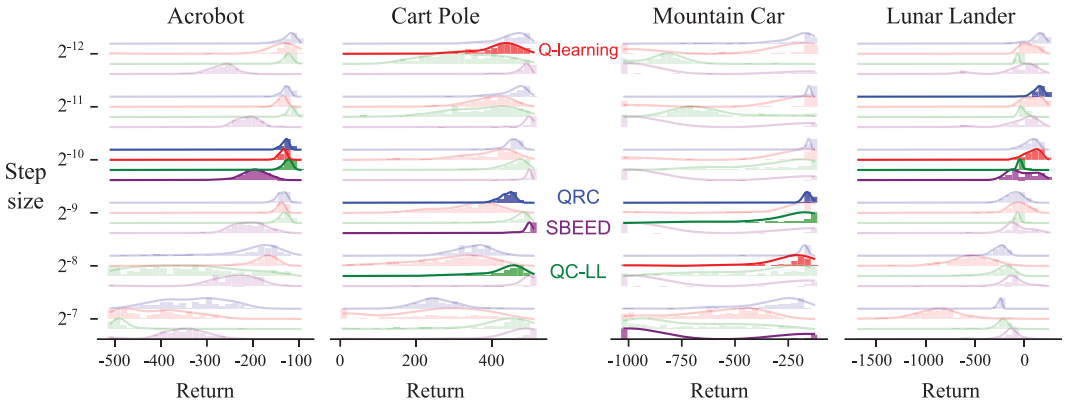

Figure 5.4: *Sensitivity to stepsize parameter.* Distribution of the return per episode for the last 25% of episodes across choice of stepsize. Each row of this figure corresponds to the performance on each algorithm across domains for one value of the stepsize parameter. Each subplot is exactly like Figure 5.3: the distribution of performance for all four algorithms using a particular stepsize parameter value on a single domain. The highlighted plots in each column represent the best performing stepsize parameter value. QRC consistently exhibits a narrow distribution of performance where the bulk of the distribution is on the upper end of the performance metric (towards the right is better). Q-learning and QC-LL both have wide performance distributions on all domains and exhibit bimodal distributions on Mountain Car. SBEED tends to exhibit bimodal performance often, with a non-trivial proportion of runs which fail to learn beyond random performance.

our experiment is run for a fixed budget of environment steps. With a smaller stepsize, QC-LL simply is not able to perform enough updates to overcome the effects of a small stepsize. This effect is exaggerated by some random influences, such as the number of steps before an agent observes its first informative reward, causing a wide range of performance across experimental trials. Nonetheless, despite the limitation of an indirect measure, these plots remain highly informative of the consistency of performance of each algorithm—which truly is what a practitioner would care about.

One way to avoid the confounding factors in Figure 5.4 is to measure stability in a more direct way. Unfortunately, there is no widely accepted definition of stability and so it is hard to propose a universal measure. In this thesis, we've typically defined stability in the dynamical system sense: a system is stable if it asymptotically approaches a fixed point or a fixed region—in many cases, this is equivalent to "convergence" of the system. Using this

idea of stability, we use the soft-divergence measure proposed in van Hasselt et al. [2018] to evaluate the internal properties of our agent in Figure 5.5. Interestingly, in the top row of subplots of Figure 5.5, we see that all three evaluated algorithms have approximately the same performance on this domain (Mountain Car). Yet the bottom row tells an entirely different story.

Let us start with the bottom-right subplot of Figure 5.5. In this plot, we measure the soft-divergence of the value function for each of three agents; QC, QRC, and Q-learning. Each solid line represents the average over 100 experimental trials and shaded regions correspond to the standard deviation. The horizontal dashed line at 100 is the magnitude of the largest observable discounted return on this domain, with discount factor $\gamma = 0.99$. We say soft-divergence occurs when the value estimate of the agent exceeds this theoretically maximal magnitude. Both of the gradient-based agents, QC and QRC, exhibit only small degrees of soft-divergence over time, while the semi-gradient method, Q-learning, quickly suffers from excessively large degrees of soft-divergence. Under this measure of stability, we would strongly conclude that semi-gradient Q-learning is highly unstable on the Mountain Car domain; while we might conclude that the gradient-based methods are both stable.

However, the bottom left subplot of Figure 5.5 suggests that soft-divergence of the values may not tell the complete story of the gradient TD methods' stability. Because gradient TD methods have a secondary internal learning process, we could likewise define a measure of instability for this process. This, however, is challenging because we have no fixed theoretical baseline of magnitude. Because we know the secondary learner is tracking the TD error, we can reason about the expected range of TD errors for this environment when an algorithm is performing stably. In Mountain Car, the agent receives a reward of -1 per timestep and uses a discount factor of $\gamma = 0.99$. From this, we'd expect that TD errors should remain around ± 1 at their largest, and naturally we would expect these decay over time. In the bottom-left subplot of Figure 5.5 we see that QC is performing quite differently than expected—the magnitude of the second weight vector is far larger than the expected 1 and this magnitude is continuing to grow at a rapid rate over time. QRC, on

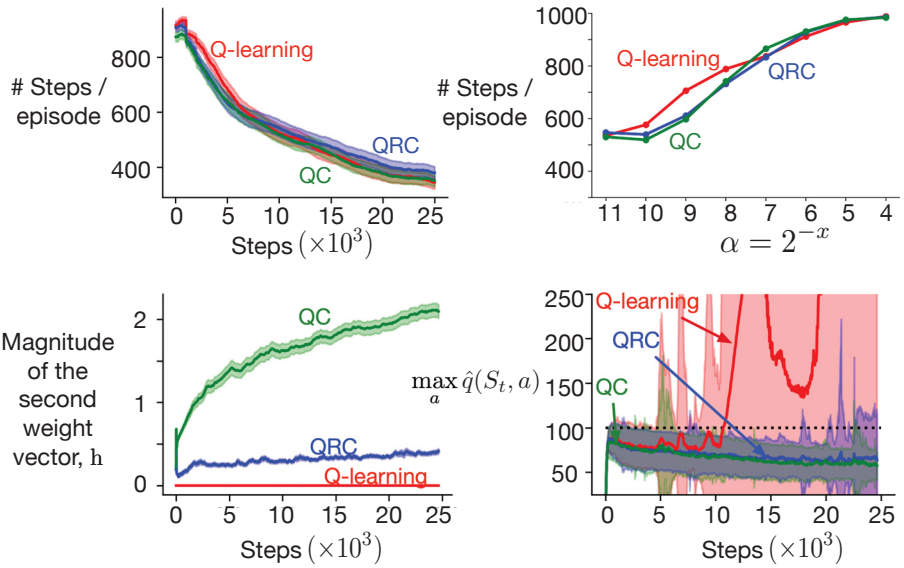

Figure 5.5: Control methods on Mountain Car with neural network function approximation. Each method takes one update step for every environment step and uses $\eta = 1$. **Top Left:** Average number of steps to goal. **Top Right:** Sensitivity to stepsize showing area under the learning curve for each value of α. **Bottom Left:** Magnitude of the secondary weights for each algorithm. Q-learning is included as a flat line at zero, as Q-learning is effectively a special case of QRC where the secondary weights are always **0**. **Bottom Right:** Mean and standard deviation of the maximum action-value for each step of learning. QC exhibited massive growth in action-values throughout learning and Q-learning exhibited periodic spikes of instability.

the other hand, maintains a fairly consistent and low-magnitude estimate over time. From this, we might conclude—unsurprisingly—that the regularization of the secondary learner helps QRC maintain stability of both internal learning processes.

5.2.3 QRC outperforms saddlepoint methods.

In this section, we revisit the decision to base QRC on a gradient-correction method as opposed to a saddlepoint method. Although we provide empirical and conceptual motivation for this choice in Section 4.4, it is worth reconfirming its validity as we move both from linear to nonlinear and from prediction to control. Recall that a primary motivation to use gradient-correction methods is that they make far less conservative updates than saddlepoint methods,

relying directly on a sample of the instantaneous TD error as opposed to a slow moving estimate of the TD error. Perhaps in nonlinear control we observe sufficient instability to justify the use of conservative updates, suggesting we would prefer to pursue saddlepoint methods. We show in this section that the less conservative gradient correction strategy remains more sample-efficient than saddlepoint methods, while retaining sufficient stability even with neural network function approximation.

We use the same approach as in Figures 5.3 and 5.4 to elucidate both the performance and stability—albeit indirectly—of two saddlepoint algorithms compared to QRC. Because we want to provide evidence to support the claim that saddlepoint methods tend to perform poorly, we need to ensure that our choice of saddlepoint update is representative. It is uninteresting to provide evidence towards this claim by deriving poor saddlepoint methods! Towards this goal, we investigate both the naive greedy GQ algorithm and a less naive GQ algorithm where the secondary learner uses the gradient of the primary learner as its feature representation; a choice supported by Section 3.2.

Unsurprisingly, in Figure 5.6, we observe that the naive saddlepoint algorithm, greedy GQ, tends to perform poorly across all environments. Counterintuitively, the performance of this algorithm typically degrades as the step-sizes increase, suggesting that the poor performance may be due to stability. Though it is not visualized here, further investigation suggests that greedy GQ here suffered from the same issue as QC in Figure 5.5: there was latent instability in the secondary learner. Because GQ relies entirely on the secondary learner to provide signal for the primary, this internal instability resulted in very slow learning performance. Unfortunately, greedy GQ cannot benefit from regularization of the secondary weights in the same way as QRC; regularizing GQ's secondary weights to be near zero inhibits the primary learner from learning, due to its sole reliance on the secondary weights to provide a learning signal!

The second saddlepoint method, GQ-Grad, uses the gradient of the primary learner as a feature representation for the secondary learner. The secondary learner, then, is a simple linear function of these features. This strategy has the

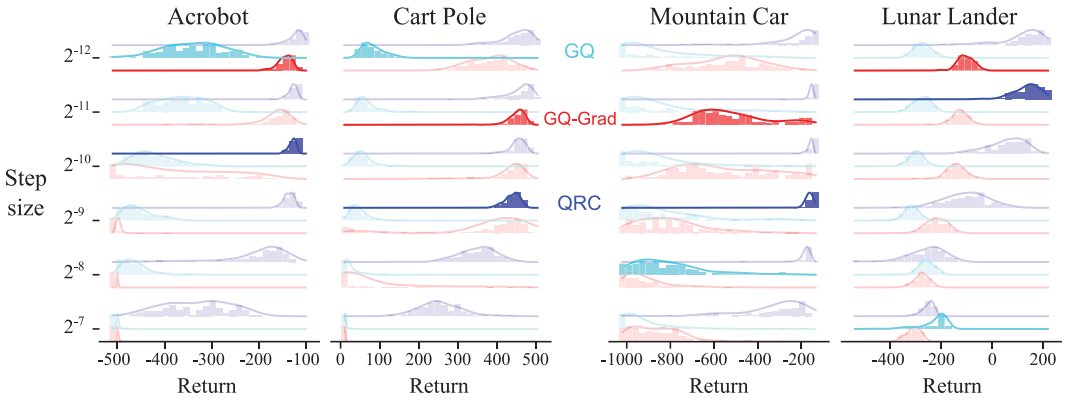

Figure 5.6: *Comparing gradient correction-based updates (QRC) and saddlepoint methods (GQ, GQ-Grad).* GQ-Grad utilizes the gradient of v as features for the secondary variable, h. Allowing the saddlepoint methods to estimate $h(s)$ by using a linear function of the gradients of the primary variable yields slightly higher performance. Nonetheless, saddlepoint methods suffer from wide performance distributions with the bulk of the distribution being further left than the gradient correction-based updates.

nice property that low magnitude gradients result in a low magnitude estimate of the expected TD error, providing a similar effect as the regularization used by QRC. Empirically, we observe in Figure 5.6 that this choice significantly improves performance over the naive greedy GQ algorithm across every domain. In two cases, Acrobot and Cartpole, this saddlepoint method performs comparable to QRC. However, we observe that the GQ-Grad algorithm still exhibits a minor degree of instability—combined with low sample efficiency—in both Mountain Car and Lunar Lander. As a result, Figure 5.6 provides evidence to support our claim that gradient-correction methods tend to outperform saddlepoint methods, even with neural network function approximation and in the control setting.

5.2.4 Using the same function class for both learners consistently performs well across environments.

In this section, we perform a final ablation of the design decisions made when combining the QRC learning rule with neural network function approximation. There is a major choice to be made, the conceptual consequences of which were of primary focus in Chapter 3. How do we approximate the secondary set of

weights for QRC?

The theory in Chapter 3 suggests three sensible choices. The first—and perhaps the most simple—is to ensure both the value function estimator and the secondary estimator share the same function approximation space. There are two ways to accomplish this, either by allowing both $\hat{Q}$ and h to have their own independent neural networks with equivalent architecture, or to make both $\hat{Q}$ and h different heads to the same neural network torso. Intuitively, we might expect this second choice to have greater sample efficiency gains at the cost of sharing function approximation resources to accomplish two possible competing goals.

The second sensible choice is to allow the secondary estimator a *larger* function class than $\hat{Q}$. In doing so, we effectively bring the $\overline{\text{PBE}^2}$ to be a closer approximation to the $\overline{\text{BE}^2}$ and in-turn protect against challenges associated with the $\overline{\text{PBE}^2}$ [Kolter, 2011]. There are again multiple ways to accomplish this. In this work, we choose to build on the shared heads approach and extend the space for h by designating some share of the hidden units in the penultimate layer of the neural network to be used only by h. The net effect is that the function class for $\hat{Q}$ is now a strict subset of that used for h and the degree of difference is controlled by the number of hidden units earmarked for h.

The final sensible choice is to use the gradients of the neural network parameters as features for the secondary head. That is, we first compute the gradient of the $\overline{\text{PBE}^2}$ for a given sample, then we learn a linear function of that gradient to track δ. Unfortunately, this approach comes with a substantial computational cost. To alleviate this cost, we make the simplification of using only the gradients of the penultimate layer of the neural network, implicitly assuming that these gradients are the most informative for controlling changes to the value function at any given instant.

To compare the effectiveness of each strategy, we use the same methodology as in the previous several subsections. We tune the configurable parameters independently for every algorithm and for every problem, then we plot the performance distribution for each level of the stepsize parameter—identifying

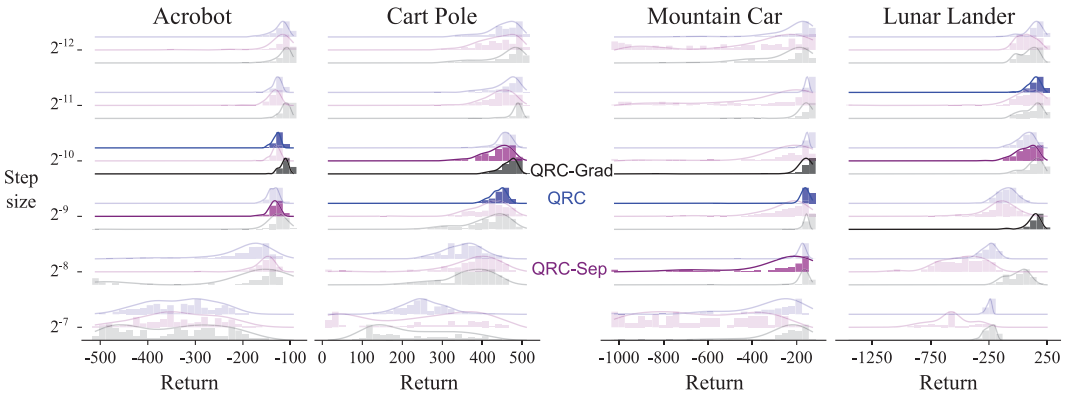

Figure 5.7: *How to represent h*: ablating the choice of basis function for the secondary variable, by comparing a shared network with two heads (QRC), two separate networks (QRC-Sep), and one network for the primary variable and a linear function of the primary variable's gradients for the secondary (QRC-Grad).

this parameter as that which best explains changes in performance. The results of this ablation are shown in Figure 5.7 with three algorithms: QRC where the secondary estimator is a second head on a shared network, QRC-Sep where both estimators have their own neural network of equivalent architecture, and QRC-Grad where the secondary estimator uses the gradients of the $\overline{\text{PBE}^2}$ for the penultimate layer.

Surprisingly, we find very little difference between methods across our four problems and even across most levels of the stepsize parameter. Perhaps the most noticeable difference between methods is the left-skew of QRC-Sep, suggesting that a small proportion of runs are performing worse than the bulk of the distribution. This aligns with our hypothesis that QRC-Sep is less sample efficient, it turns out these lower performing runs were still improving performance at the time that the experiment was terminated due to limiting the experiment to a small number of environment interactions. Comparing the differences in mean between each of the best performing configurations, we find no detectable, statistically significant difference between methods. This is actually a freeing finding; all three are fine choices, allowing us the freedom to pick that which is most conceptually simple and computationally efficient.

5.3 Reliability and empirical science

The reliability and empirical science of reinforcement learning systems are deeply intertwined. An unreliable system is difficult to study empirically, requiring a large number of samples and sophisticated analytical techniques to deal with the variation in results [Colas et al., 2018, Agarwal et al., 2021, Patterson et al., 2023]. Empirical methodologies designed for reliable systems, but applied to unreliable systems, produce irreplicable conclusions and fallible insights [Islam et al., 2017, Henderson et al., 2018, Patterson et al., 2023].

The predominant approach to breaking this cycle has been to design and recommend empirical strategies which are better suited for studying unreliable, high-variance systems. The natural starting point to improving our statistical insights is to simply collect more data; it is impossible to determine if a system is reliable if that system has not been run sufficient times to track variation in performance. Many studies have highlighted the critical need to collect more data, making appeals to the assumptions underlying statistical tests [Colas et al., 2018, Machado et al., 2018, Colas et al., 2019], showing wide confidence regions about the mean [Henderson et al., 2018, Jordan et al., 2020, Agarwal et al., 2021, Patterson et al., 2023], or even directly describing the unreliability of empirical conclusions [Islam et al., 2017, Sajed et al., 2018, Pineau et al., 2020, Patterson et al., 2023].

Complementing the call for increased data, several studies have designed better empirical methodologies and statistical analysis techniques to handle the varied data stemming from unreliable learning systems. Much of the motivation is to reconcile the high cost of obtaining more data with improved analysis. One proposed approach is to throw out half of the data, that belonging to the best instances and the worst instances equally, and report the average over the remaining samples [Agarwal et al., 2021]. Confidence intervals about this statistic, the interquartile mean (IQM), have been shown to be tight and conservative. Another complementary approach is the use of bootstrapped statistics to compute distribution-agnostic confidence intervals, recognizing that unstable learning systems tend to produce non-normal per-

formance data [Colas et al., 2019, Henderson et al., 2018, Jordan et al., 2020, Agarwal et al., 2021, Patterson et al., 2023]. Finally, strategies for combining data from multiple environments to form more robust statistics have been shown to reduce the total empirical cost, while producing reliable conclusions [Whiteson et al., 2011, Jordan et al., 2020, Patterson et al., 2023].

In this section, we will explore the relationship between reliable algorithms and reproducible empirical research. We will discuss statistical analysis techniques to evaluate the reliability of reinforcement learning systems, using commonly collected performance data. We will show that hyperparameter configuration plays a large role in the reproducibility of empirical research, and that unreliable learning systems add a degree of complexity to this challenge. Throughout these discussions, we will primarily compare the QRC algorithm to the DQN algorithm [Mnih et al., 2013], showing that QRC is a prototypical example of a reliable learning algorithm.

Evaluating reliability. Early in this thesis we defined a *reliable* learning system as a system that produces individual agents that consistently achieve similar performance, despite different random influences on each individual agent. In this way, reliability is focused on individual agents behaving similarly, even when each agent starts from different initializations and experiences different streams of experience. By contrast, conventional empirical methodologies seek to understand differences in populations of agents often by comparing a population-level statistic between two populations, such as their average performance.

In this section, we start by illustrating why population-level statistics can be misleading and how increasing the number of samples can exaggerate this problem. We show that, despite recent attention to improved confidence interval techniques, uncertainty metrics do not resolve this issue. We highlight that even the most naive strategy of evaluating every agent individually can help avoid being misled by aggregate statistics. Then we conclude with a proposed aggregate statistic, the *tolerance interval*, that captures both individual variation and uncertainty, making this a powerful tool for scientifically

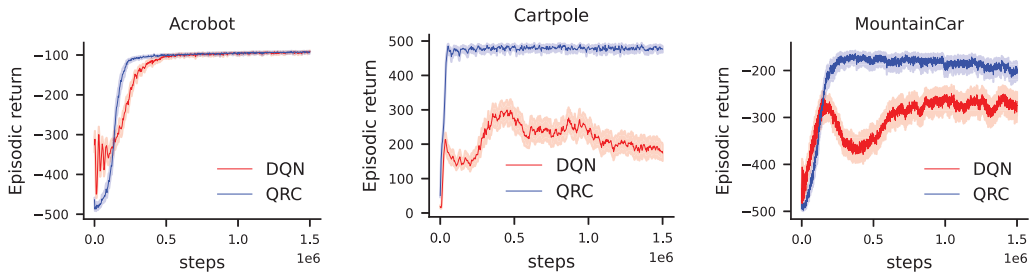

Figure 5.8: Average performance over 200 agents with 95% bootstrap percentile confidence intervals represented as a shaded region about the mean.

understanding the reliability of learning systems.

The fallacy of the average agent. Let us revisit the experimental methodology of Section 5.2. We will focus on three classic control environments: Acrobot, Cartpole, and Mountain Car. For now, we will assume that we have well-configured both algorithms of interest—QRC and DQN—choosing hyperparameter settings for each individual environment that maximize the performance of each of these algorithms after 200k steps of interaction. Specifically, we will use the best configuration found from the experiments in Section 5.2. Unlike Section 5.2, however, we will now run every agent for 1.5M steps of interaction, or roughly 7 times longer, and we will run both algorithms 200 times each, generating 200 individual agents for each environment.

We report the average performance of each agent in Figure 5.8. With 200 individuals for every condition—algorithm and environment—the average performance curves are relatively smooth, and the confidence intervals are quite small. In fact, the confidence intervals are barely visible in all three cases. Certainly, it is clear that DQN performs poorly on average in two of the three environments, however, we might wrongly conclude that both QRC and DQN are consistent in all three environments.

Based on Figure 5.8, we would misleadingly—but correctly—claim that DQN achieves approximately 250 return on average in Cartpole and -300 return on average in Mountain Car for a bulk of the learning curve. If instead we investigate the individual performances of these agents in Figure 5.9, we find that individual agents produced by the DQN algorithm are often sporadic

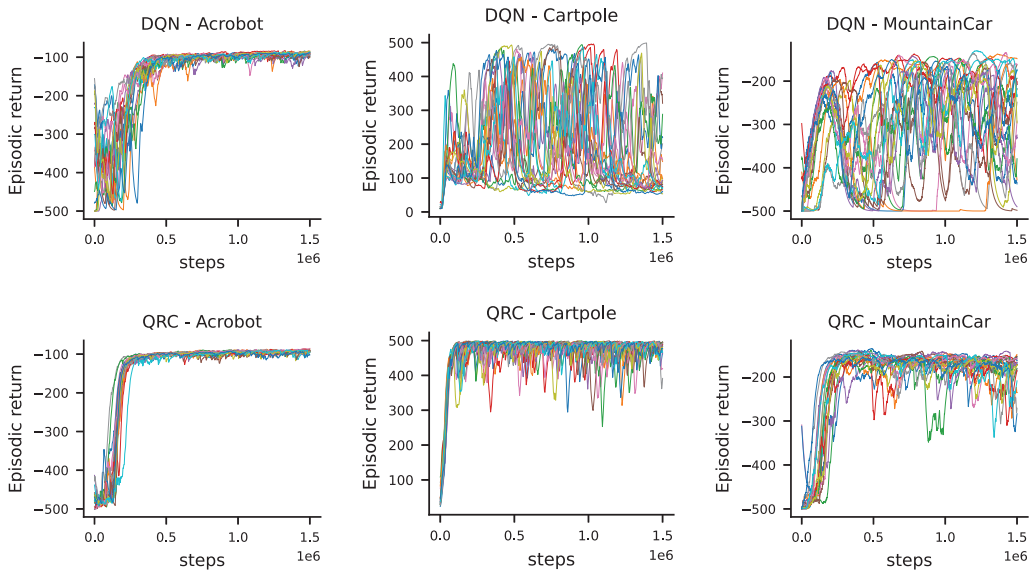

Figure 5.9: Individual performance of 20 randomly selected individual agents for both DQN (top row) and QRC (bottom row).

with some agents performing well over time, other agents performing poorly over time, and most agents bouncing between these two. QRC, on the other hand, consistently produces agents that perform well.

An important takeaway from Figures 5.8 and 5.9 is that unreliability is not an inevitability. While it is important to develop improved analytical strategies and empirical methodologies to study unreliable algorithms such as DQN, it is possible to simultaneously improve the reliability of our methods through further algorithm development. Both learning systems, in this case, are highly related with the only difference being the update rule. Both algorithms were configured with the most naive configuration strategy, grid search, where DQN had the advantage of tuning an additional configuration parameter, the target network refresh rate.

Tolerance intervals. Unfortunately, the plots in Figure 5.9 are incredibly difficult to compare quantitatively. It is clear qualitatively that there is a difference between QRC and DQN, but these "spaghetti" plots obfuscate several details. Perhaps the most notable limitation is that these plots rely on a subsample of the data, only 20 out of 200 agents are represented. What

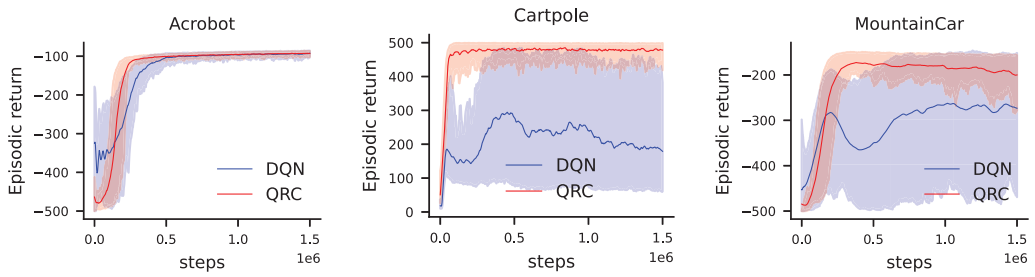

Figure 5.10: Tolerance intervals with $\alpha = 0.05$ and $\beta = 0.9$ for both DQN and QRC. For context, the mean performance across individuals is also shown as a solid line for both algorithms.

if, when investigating all 200 individual agents, some instances of QRC also perform sporadically? Or perhaps a majority of DQN's individual agents are entirely consistent, but the limitations of the plotting mechanism exaggerate a few particularly chaotic individuals?

To address these concerns, we can use a form of statistical interval called a *tolerance interval* to provide an aggregate summary of the individual agents. Similar to a confidence interval, the tolerance interval allows us to reason about our uncertainty in our data; *what if we collected more individual agents, would our conclusions change?* Unlike a confidence interval, the tolerance interval captures the underlying variation across the individuals in our sample, much like a *prediction interval*. The tolerance interval answers the question: *Given a* $1-\alpha$ *confidence level, what range of performance values captures* β *proportion of my individual agents?* Concretely, a $(\alpha = 0.05, \beta = 0.9)$ tolerance interval captures 90% of individual performance values with 95% confidence.

In Figure 5.10 we show the $(\alpha = 0.05, \beta = 0.9)$ tolerance interval for both DQN and QRC across our three classic control domains. As we might expect from Figure 5.9, QRC's tolerance interval is tight across all three environments suggesting that at least 90% of the time, an agent produced by the QRC algorithm will perform near-optimally on these environments. Similarly, on Acrobot, the tolerance interval for DQN suggests that DQN reliably produces agents that solve this domain. On Cartpole and Mountain Car, however, we are unable to predict ahead of time how an agent produced by DQN will perform. In fact, the tolerance interval indicates that a DQN agent could

perform anywhere from the worst possible performance to the best possible performance in either domain. From Figure 5.10, we can strongly conclude that DQN is not a reliable algorithm on Cartpole and Mountain Car.

5.4 Summary

In this chapter, we introduced the Q-learning with regularized corrections algorithm (QRC), which is a simple modification of the TDRC algorithm studied in Chapter 4. We described a strategy to use the QRC algorithm alongside neural network function approximation for the problem of off-policy control, by embedding this update rule in a DQN-like learning system, replacing the q-learning update rule and obviating the need for target networks. We empirically validate the performance of this control algorithm by comparing idealized performance with respect to configuration parameters (stepsizes, buffer parameters, etc.) of SBEED [Dai et al., 2018], DQN [Mnih et al., 2013], and nonlinear QC [Maei et al., 2009]. Finally, we investigate the reliability of each of these methods by investigating the empirical distribution of their performance over repeated runs, capturing variation due to initial conditions and stochasticity in the data-generating process.

In the next chapter, we will revisit the biconjugate objectives defined in Chapter 3 and use this reformulation to propose two new statistically robust biconjugate objectives. We will then use the insights from TDRC and QRC to derive algorithms to minimize these robust objectives, then empirically investigate their performance compared to QRC and DQN.

Chapter 6

Statistically Robust Bellman Errors

In this chapter, we introduce a robust Bellman objective which allows us to control the tradeoff of high magnitude errors across states. We will build on the biconjugate reformulation of the Bellman error established in Chapter 3, allowing us to derive algorithms with provable stability guarantees under broad conditions. The algorithms derived in this chapter are built on TDRC and QRC from Chapter 4, inheriting TDRC's reliability by being gradient-based methods. The algorithms introduced in this chapter achieve an even greater degree of reliability through the use of novel statistically robust Bellman objectives.

Many algorithms in reinforcement learning are built on objectives that use squared errors. Squared errors, however, tend to magnify incorrect predictions; encouraging the function approximator to expend limited representation resources on states and actions that induce the highest Bellman error. High magnitude Bellman errors can occur during the learning process and can persist at convergence due to state aliasing or irreducible noise in the observations.

This focus on large Bellman errors can be particularly undesirable in control. As an example, consider the CliffWorld domain. The agent starts in one corner of a grid and seeks to walk alongside a cliff to reach the opposite corner on the same wall. If the agent steps into the cliff, it receives a high-magnitude negative reward and must start again. The agent otherwise receives -1 reward per step until it reaches the goal and the episode terminates. In order to

successfully solve this domain, the agent need only learn that actions which step into the cliff yield more negative return than actions which step toward the goal. Representing the exact magnitude of this expected negative return is unnecessary. Squared errors do just the opposite: by squaring large errors, they expend their limited representation capacity focusing on those states. As a result, the representation may suffer for other states and actions providing a suboptimal ordering over actions. Further, during the optimization, these high-magnitude errors can introduce transient instability in the learning process.

This challenge has been addressed heuristically through a variety of approaches in RL, which include clipping rewards [Hessel et al., 2018b], errors [Mnih et al., 2013], and gradients [Van Hasselt et al., 2016]; careful manipulation of the reward function [Brockman et al., 2016, Young and Tian, 2019]; and variance reduction methods [Wang et al., 2016, Hessel et al., 2018a]. Some approaches, such as manipulating the reward function, require extensive domain knowledge and are not generally possible for all problem settings. Other approaches, such as clipping, inhibit analyzing the fixed-point of the update. Error clipping in Q-learning algorithms—often referred to as minimizing a Huber loss—is built on a semi-gradient update which does not follow the gradient of any loss function. This makes analysis of the fixed-point challenging, leaving open the question: what is the effect of clipping the error on TD-like algorithms?

Huber-like losses have become commonplace in deep reinforcement learning implementations [Raffin et al., 2021, Dhariwal et al., 2017, Mnih et al., 2013]. Although these implementations often claim to be minimizing a Huber loss, the iterative weight update does not minimize any known loss function. They apply the Huber function to the TD error, $p_\tau\left(R_{t+1} + \gamma_{t+1} v_{w_{\text{old}}}(S_{t+1}) - v_w(S_t)\right)$, with a fixed target network $v_{w_{\text{old}}}$. This update more closely resembles that of a mean Huber TD error, rather than Bellman error, because the Huber is inside the expectation: $\mathbb{E}_\pi\left[p_\tau(\delta_t) \mid S_t = s\right] \neq p_\tau\left(\mathbb{E}_\pi\left[\delta_t \mid S_t = s\right]\right)$. Further, the target network causes the objective to change with time. Analytically, the fixed-point of this DQN update remains an open question. Empirically, clipping

the magnitude of the TD errors has led toward better learning stability on certain problem settings [Mnih et al., 2013], however this insight is inconsistent when validated across a wider test bed of problem settings [Obando-Ceron and Castro, 2021]. By instead defining a Huber BE objective, we can concretely characterize the loss surface and solutions of the proposed objective.

6.1 The mean Huber Bellman error

In this section, we define the mean Huber Bellman error, $\overline{\text{HBE}}$, as well as the mean absolute Bellman error, $\overline{|\text{BE}|}$. The $\overline{\text{HBE}}$ depends on a configuration parameter τ which allows the Huber error ($\overline{\text{HBE}}$) to smoothly interpolate between the squared error ($\overline{\text{BE}^2}$) and the absolute error ($\overline{|\text{BE}|}$). All three Bellman errors share the same fixed-point in the tabular setting—and more generally when v_π belongs to the parameterized function class—but can have notably different fixed-points under limited function approximation.

The $\overline{\text{HBE}}$ is straightforward to specify,

$$\overline{\text{HBE}}(w) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} d(s) p_\tau \left(\mathbb{E}_\pi \left[\delta(w) \mid S = s \right] \right) \tag{6.1}$$

where $p_\tau(\cdot)$ is the Huber function [Huber, 1964],

$$p_\tau(x) \stackrel{\text{def}}{=} \begin{cases} x^2 & \text{if } |x| \leq \tau \\ 2\tau|x| - \tau^2 & \text{otherwise} \end{cases}$$

for some parameter $\tau > 0$. In a region about the origin, the size of which is controlled by τ, the Huber function behaves as the square function. Outside this region, the Huber function becomes the absolute value. The threshold parameter, τ controls the inflection point between the square and absolute functions. Similarly, we can define the $\overline{|\text{BE}|}$:

$$\overline{|\text{BE}|}(w) \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} d(s) |\mathbb{E}_\pi \left[\delta(w) \mid S = s \right]|. \tag{6.2}$$

Both robust objectives, as well as the $\overline{\text{BE}^2}$, rely on an average over states, weighted according to distribution d. Typically, this distribution is the steady-state distribution of the MDP when following the behavior policy; that is

$d = d_b$. However, as we saw in Chapter 3, this is not the only choice of state distribution that we can make.

Implicitly, the statistic of the residuals that we choose to measure—the Huber function, the absolute function, or the square function—also impacts the effective weighting over states. The square function increases the effective weighting proportional to the magnitude of the residual; if the error in a given state is large, then that state is given more effective weight in the total loss. By contrast, the absolute function gives no additional effective weighting to a state based on the magnitude of its error. Naturally, the Huber function interpolates between these two based on its threshold parameter.

Although the $\overline{\mathrm{HBE}}$ is straightforward to specify, it is not necessarily straightforward to optimize. The difficulty is obtaining a sample of the gradient of this objective for the same reason as the $\overline{\mathrm{BE}^2}$: the *double sampling* issue. Due to the chain rule and the nonlinearity of $|\cdot|$ and $p_\tau(\tau)\cdot$, both the $\overline{|\mathrm{BE}|}$ and $\overline{\mathrm{HBE}}$ will suffer from the same issue as the $\overline{\mathrm{BE}^2}$.

To facilitate optimizing the $\overline{\mathrm{HBE}}$ and $\overline{|\mathrm{BE}|}$, we reformulate the objectives using biconjugates using the same strategy as in Chapter 3. Because the three functions we want to reformulate—the absolute, Huber, and square functions—are all proper, convex, and lower semi-continuous, this equivalence allows us to reformulate these losses using biconjugates to avoid the double sampling issue without changing the solutions to these losses or even the loss surface.

The absolute value has a well known biconjugate $\max_{h\in[-1,1]} xh$. It is easy to see that this biconjugate is indeed equivalent to the original function, that is $|x| = \max_{h\in[-1,1]} xh$. Whenever the input x is positive, the maximization is realized at $h = 1$ and so the biconjugate takes value x. Whenever the input x is negative, the maximization is realized at $h = -1$ and so the biconjugate again takes value x.

The biconjugate for the Huber function is less readily available. Though it is a relatively straightforward result to obtain, to the best of our knowledge, it is new and so worth providing formally. We derive the biconjugate form for the Huber error in the following proposition.

Proposition 3 *The biconjugate of the huber function is* $f_\tau^{**}(x) = \max_{h\in[-\tau,\tau]} xh - \frac{1}{2}h^2$.

We provide a proof of this proposition in Appendix B.1.

Like the Huber function itself, the biconjugate of the Huber function adopts properties from both the square and absolute value functions. The Huber function's biconjugate differs from the squared function's biconjugate only in its use of a constrained optimization. These constraints are similar to those used by the absolute value function, though in the case of the Huber function they are inequality constraints.

We can now provide the biconjugate forms for $\overline{|\text{BE}|}$ and $\overline{\text{HBE}}$:

$$\overline{|\text{BE}|}(w) \stackrel{\text{def}}{=} \max_{h\in\mathcal{H}_{\text{sign}}} \sum_{s\in\mathcal{S}} d(s)h(s)\Delta(s)$$
$$\overline{\text{HBE}}(w) \stackrel{\text{def}}{=} \max_{h\in\mathcal{H}_{\text{clip}}} \sum_{s\in\mathcal{S}} d(s)(2h(s)\Delta(s) - h(s)^2)$$

$\mathcal{H}_{\text{sign}}$ is the set of all functions $h_{\text{sign}} : \mathcal{S} \to \{-1, 1\}$ and $\mathcal{H}_{\text{clip}}$ the set of all functions $h_{\text{clip}_\tau} : \mathcal{S} \to [-\tau, \tau]$. Notice that for both the $\overline{\text{HBE}}$ and $\overline{|\text{BE}|}$, we have a constrained optimization problem for h, which differs from the $\overline{\text{BE}^2}$. It is also worth noting that, like the $\overline{\text{BE}^2}$, the biconjugate form of the $\overline{|\text{BE}|}$ and $\overline{\text{HBE}}$ depend on functions whose domain are the underlying states of the MDP, $\mathcal{S}$. Similar to the $\overline{\text{BE}^2}$, this leads to a contradiction: the function h can use the underlying states to evaluate the agent, while the value function is limited to functions of the observations $\mathcal{O}$. We will resolve this limitation, creating identifiable versions of the $\overline{|\text{BE}|}$ and $\overline{\text{HBE}}$ in Section 6.2.

To gain some intuition on how these objectives differ, consider a two-state MDP where both states are aliased with a single observation. The agent starts in the first state and deterministically transitions to the second state, where the agent remains with high probability or terminates the episode with low-probability. Because the length of each episode varies greatly and because the first state is visited infrequently, the distribution of TD errors becomes skewed by large errors in the first state. Updating the prediction to decrease the high error in the first state harms the prediction in the second state due to the state

aliasing. Because the true value function is not representable, the minimizer of each objective must trade off prediction error in each state.

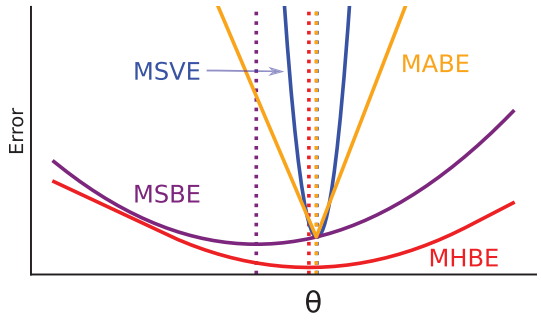

Figure 6.1: Objectives and fixed-points on the above described MDP. Dotted lines are drawn at the minima. The fixed-points of the robust objectives are much better proxies for the $\overline{\text{VE}}^2$. Note that the $\overline{|\text{VE}|}$ (not shown) has a similar fixed-point to the $\overline{\text{VE}}^2$, so the $\overline{\text{HBE}}$ and $\overline{|\text{BE}|}$ also well approximate the fixed-point of the $\overline{|\text{VE}|}$.

Figure 6.1 visualizes this trade-off among objectives. Surprisingly, even when our goal is to minimize a mean *squared* VE, the mean *absolute* BE provides the closest solution among the Bellman objectives. The $\overline{\text{HBE}}$ provides a close approximation as well as a better optimization surface. The $\overline{\text{BE}}^2$, on the other hand, defines a poor fixed-point, highly skewed by large errors in the first state of the MDP.

Differences from supervised learning. In both supervised learning and reinforcement learning, we learn some statistic of our targets that is conditioned on our input data. Classically, in supervised learning, this might be $\mathbb{E}[y \mid x]$ for inputs x and target y. In supervised learning, robust losses are used to mitigate issues with high-variance, stochastic targets. With the absolute loss producing an unbiased conditional median estimator that is highly robust to outlier values, and the Huber function providing a more generalized M-estimator.

In the reinforcement learning setting, the target for Bellman errors is itself an expectation and has no variance. Instead, the robust Bellman losses impact the accumulation of error across states. In the tabular setting, or generically when v_π is realizable, all three Bellman errors share the same solution: the true

expected return. That is, with sufficient function approximation capacity, the choice between robust losses and the squared loss does not impact the learned solution. The same is not true of supervised learning; using a robust loss fundamentally changes the statistic being estimated, regardless of approximation capacity.

Another notable deviation from supervised learning is the nested expectations found in Bellman errors. These nested expectations lead to a double sampling issue in all three objectives, where gradients of the objective are composed of a product of expectations. This product of expectations does not appear in supervised learning, allowing supervised learning to take stochastic gradients through the robust losses trivially. In reinforcement learning, we instead must find some alternative strategy for obtaining gradient samples for the robust losses. In the previous section, we achieve this by reformulating the robust objective through the use of biconjugates, as we did with the squared loss in Chapter 3.

6.2 Limiting the function class of the Huber Bellman error

In practice, we will generally have parameterized functions v and h, and so the biconjugate objectives will no longer perfectly obtain the maximum $h^*(s)$. One source of error is from approximation due to limited computation and a finite number of samples. Another source of error is due to limitations of the chosen parameterized function class. This limitation of the function class can actually be seen as a projection on the Bellman errors, previously highlighted for the $\overline{\text{BE}^2}$ [Patterson et al., 2022b] and in Chapter 3, which we show in this section for the $\overline{\text{HBE}}$.

In the finite state setting, we can represent the parameterized function $h \in \mathcal{H}_\theta$ as a vector $u \in \mathbb{R}^{|\mathcal{S}|}$ composed of entries $\mathbb{E}_\pi \left[\delta(S) \mid S = s\right]$; thus the vector $u = \mathcal{T} v_w - v_w$. The orthogonal projection of x onto $\mathcal{H}_\theta$ is defined as

$$\Pi_{\mathcal{H}} x \stackrel{\text{def}}{=} \arg\min_{h \in \mathcal{H}_\theta} \|x - h\|_d$$

for convex subset $\mathcal{H}_\theta$, where $d : \mathcal{S} \to [0, 1]$ is a weighting over states. For the Huber loss, we further restrict $\mathcal{H}_\theta$ to get $\mathcal{H}_{\text{clip}} = \mathcal{F}_{\text{clip}} \cap \mathcal{H}_\theta$. Because $\mathcal{F}_{\text{clip}}$ is convex and contains $\mathbf{0}$ in its interior for $\tau > 0$, because $\mathcal{H}_\theta$ is convex by assumption, and because the intersection of convex sets is convex, then there exists a projection, $\Pi_{\mathcal{H}_{\text{clip}},d}$.

Despite changing the projection from $\Pi_{\mathcal{H}}$ for the $\overline{\text{PBE}^2}$ to $\Pi_{\mathcal{H}_{\text{clip}}}$ for the $\overline{\text{HPBE}}$, both objectives share the same fixed-point when $\mathcal{H} = \mathcal{V}$, because the TD fixed-point has zero $\overline{\text{PBE}^2}$ and further projection has no effect.

Theorem 4 *Let $\mathcal{V}$ be a convex set of functions, v_w, where $\overline{PBE^2}(w) \stackrel{def}{=} \|\Pi_{\mathcal{V},d}(\mathcal{T}v_w - v_w)\|_d^2$. Let $\mathcal{F}_{clip,v} \stackrel{def}{=} \mathcal{F}_{clip} \cap \mathcal{V}$. Then the solution is the same for the $\overline{PBE^2}$ and the $\overline{HPBE}$ when $\tau > 0$. Further, this solution has zero error under both objectives.*

Proof: Let w^* be the solution to the $\overline{\text{PBE}^2}$. Then

$$\begin{aligned}
\overline{\text{HPBE}}(w^*) &= \|\Pi_{\mathcal{F}_{\text{clip}},v}(\mathcal{T}v_{w^*} - v_{w^*})\|_d^2 \\
&\leq \|\Pi_{\mathcal{F}_{\text{clip}},v}\Pi_{\mathcal{V}}(\mathcal{T}v_{w^*} - v_{w^*})\|_d^2 \\
&\leq \|\Pi_{\mathcal{V}}(\mathcal{T}v_{w^*} - v_{w^*})\|_d^2 = 0
\end{aligned}$$

The first inequality follows from the fact that projecting onto the set $\mathcal{V}$ first and then projecting further to $\mathcal{F}_{\text{clip}}$ cannot have a smaller norm than projecting directly to $\Pi_{\mathcal{F}_{\text{clip}},v}$. The second inequality follows from the fact that the projection is a contraction under d, again by definition.

Further, for any w when $\overline{\text{PBE}^2}(w) > 0$ then $\overline{\text{HPBE}}(w) > 0$. We show this by contradiction. Assume that for some w, $\overline{\text{PBE}^2}(w) > 0$ and $\overline{\text{HPBE}}(w) = 0$.

Let $\Delta = \mathcal{T}v_w - v_w$, then

$$
\begin{aligned}
\overline{\text{HPBE}}(w) =& \|\Pi_{\mathcal{F}_{\text{clip}},v}\Delta\|_d^2 = 0 && \triangleright \text{ By assumption} \\
\implies & \Pi_{\mathcal{F}_{\text{clip}},v}\Delta = \mathbf{0} && \triangleright \text{ Positive definiteness of norm} \\
\implies & \Delta \in \ker(\Pi_{\mathcal{F}_{\text{clip}},v}) && \triangleright \text{ Definition of kernel} \\
\implies & \Delta \in \mathcal{N}(\mathcal{V}) && \triangleright \ \ker(\Pi_{\mathcal{F}_{\text{clip}},v}) = \mathcal{N}(\mathcal{V}) \\
\implies & \Pi_{\mathcal{V}}\Delta = \mathbf{0} && \triangleright \ \Delta \in \ker(\Pi_{\mathcal{V}}) \text{ because } \Delta \in \mathcal{N}(\mathcal{V}) \\
\implies & \|\Pi_{\mathcal{V}}\Delta\|_d^2 = 0 \\
\implies & \overline{\text{PBE}^2}(w) = 0,
\end{aligned}
$$

yielding a contradiction. The fourth step is because clipping has no effect on the nullspace, $\mathcal{N}(\mathcal{V})$, so any vector that $\Pi_{\mathcal{F}_{\text{clip}},v}$ projects to $\mathbf{0}$ is in the nullspace $\mathcal{N}(\mathcal{V})$. Specifically, because $\mathcal{F}_{\text{clip},v} \subseteq \mathcal{V}$ and the only vector u that satisfies $\langle u, v\rangle = 0$ for all $v \in \mathcal{F}_{\text{clip}}$ is the vector $u = \mathbf{0}$, then $\ker(\Pi_{\mathcal{F}_{\text{clip}},v}) = \{\mathbf{0}\} \cup \ker(\Pi_{\mathcal{V}})$ and the kernel of a linear map is the nullspace of its domain, which already includes $\mathbf{0}$. □

This connection gives insight into why TDRC has empirically demonstrated better stability, despite being designed to optimize the $\overline{\text{PBE}^2}$—a squared error. The regularization term in TDRC acts as a soft constraint on h, biasing h towards low-magnitude values. In fact, this connection is identical to the connection between ℓ_2 regularization and the Lagrangian for certain linear constraints. We visualize the connection in Figure 6.2.

6.3 Empirical analysis of the fixed-points

Before developing online learning algorithms to minimize the robust Bellman objectives, we first seek to empirically understand the idealized solutions of these objective functions in relationship to the squared (projected) Bellman error. The intuition established in Section 6.1 tells us we should expect statistical robustness to yield better minimizers in challenging problems with a large degree of state aliasing. In this section, we design a set of exemplary toy problems with exaggerated characteristics that highlight differences between

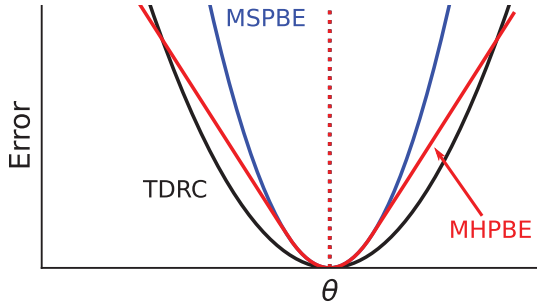

Figure 6.2: Visualizing the loss surface for the $\overline{\text{PBE}^2}$ (blue), the $\overline{\text{HPBE}}$ (red), and an approximation of the loss followed by the TDRC algorithm (black). TDRC does not define a valid projection, but we can compute the ℓ_2 regularized solution for h for each θ, to plot the idealized loss surface. The $\overline{\text{HPBE}}$ is a squared projected loss, but the projection under the Huber flattens the surface for large residuals, characteristic of the flat regions of the Huber function. TDRC has a less sharp surface for a local region near the fixed-point, but ultimately suffers from using a squared loss for very large residuals.

objectives. Our goal is not to show that any one objective necessarily dominates the others, but rather to highlight differences between objectives via highly controlled and manipulated problems. Later in Section 6.5, we will investigate algorithms that optimize these robust objectives on more commonly used benchmarks.

The empirical methodology in this section follows an exploratory format, as opposed to more classic hypothesis testing. Instead of fixing a set of problems and evaluating solution strategies on those fixed problems, we are fixing the solution strategies and probing them with a different set of problems. As a result, the learning problems proposed in this section are highly cherry-picked, each custom-designed with a particular property in mind. We pose the following research question: *"For which set of problem properties does each objective provide a favorable solution?"*

The problems. For each of the following problems, we assume that the state is fully observable, and the agent constructs some features of this state, say by a neural network or tile-coding. This feature generating function is fixed and provided to the agent; for now, we only study the linear mapping from features to values. This models a common setting where the agent receives a

complete Markov state, learns a feature generating function from those states, then learns a linear value function from those features—e.g. using an end-to-end neural network learning procedure such as DQN. Although the agent has access to the true underlying state, we cannot assume that the feature generating function will always yield useful features which cleanly discriminate between states of highly different value, especially early in the learning process.

The first two problem settings build on simple MDPs that have state representations which aggressively alias multiple states into a single feature. Because this aliasing skews the TD errors, we expect to find that the robust objectives perform well compared to the squared Bellman error. **HardAlias-1** is an 8-state random walk where the first, third, and final states share a common feature, and the remaining five states share three features. **HardAlias-2** is the 2-state problem from Tsitsiklis and Van Roy [1997], which was originally designed to highlight the insufficiency of minimizing the squared Bellman Residual, with lightly modified reward so the optimal value function cannot be perfectly represented.

The next investigated problem setting (**Outlier**) we design to highlight the advantage of the $\overline{\text{HBE}}$ by creating a single outlier state with a large magnitude return among a large set of states with approximately normally distributed returns. We use a randomly initialized frozen neural network to generate five features with a one-hot state encoding as input. The agent starts in a state that has an $\epsilon = 0.01$ chance of terminating immediately with -1000 reward, or a $1 - \epsilon$ chance of entering the middle state of a 49-state random walk.

The final two problems are chosen to highlight a scenario where the $\overline{\text{BE}^2}$ finds favorable solutions compared to the robust objectives, which we expect will behave more conservatively in this idealized setting. In these problems, the returns are distributed approximately normally across states and states are lightly aliased. We use two random walks, the first with $N = 5$ states (**SmallChain**) and the other with $N = 19$ states (**BigChain**), with a randomly initialized neural network representation of size $\frac{N}{2}$. The agent receives a reward of -1 or $+1$ on the left and right-most states respectively.

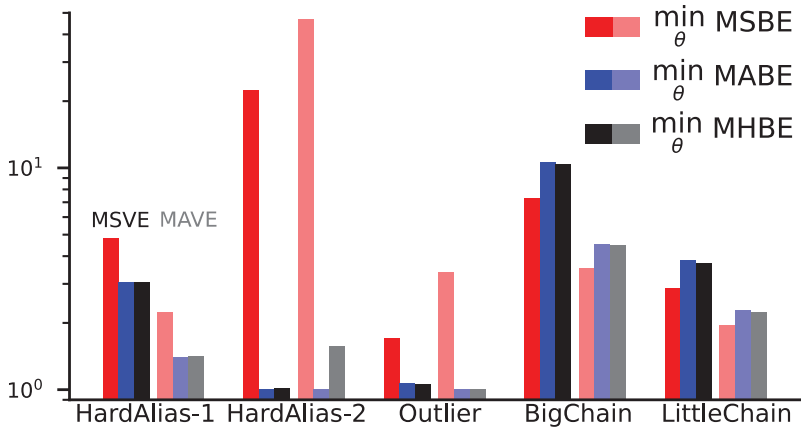

Figure 6.3: Evaluating the quality of the fixed-points of each objective function according to the $\overline{\text{VE}^2}$ and $\overline{|\text{VE}|}$ across several prediction problems. Error is plotted relative to the best representable value function. The robust losses are better in the hard aliasing domains, the $\overline{\text{HBE}}$ is slightly better in Outlier, and the $\overline{\text{BE}^2}$ is better on the classic random walks.

Analyzing the fixed-points. Figure 6.3 shows the $\overline{\text{VE}^2}$ and $\overline{|\text{VE}|}$ for each fixed-point, relative to the best realizable value function on each problem. That is, a value of 10 on the y-axis means that the solution to the given objective is 10 times worse than the best realizable solution using the same set of features. This gives a measure of how good of a proxy each Bellman objective is to the $\overline{\text{VE}^2}$ (bold colors) or $\overline{|\text{VE}|}$ (faded colors).

One very clear outcome of this analysis is that the $\overline{\text{BE}^2}$ can define shockingly poor solutions in the case of harmful state-aliasing. Contrarily, it was surprisingly difficult to find settings where the $\overline{\text{BE}^2}$ performed favorably to the same extreme compared to the robust objectives. We additionally found that it was difficult to design a problem setting where the $\overline{\text{HBE}}$ outperformed both the $\overline{\text{BE}^2}$ and $\overline{|\text{BE}|}$. Perhaps this should not be surprising, the $\overline{\text{HBE}}$ smoothly interpolates between the others for varying values of its configuration parameter, τ. While the $\overline{\text{HBE}}$ does manage to outperform both the $\overline{\text{BE}^2}$ and $\overline{|\text{BE}|}$ on the Outlier problem, the degree of difference compared to the $\overline{|\text{BE}|}$ is negligible.

A final important takeaway from Figure 6.3 is that both robust objectives defined fixed-points that were consistently good. On the hard aliasing problems, the robust objectives were indeed robust to the extreme deviations in the TD error; in the case of HardAlias-2, the solution to the $\overline{|\text{BE}|}$ outperformed the

solution to the $\overline{\text{BE}^2}$ by more than twenty times! In the more benign problems where the solution to the $\overline{\text{BE}^2}$ shined, the solutions to the robust objectives were only fractionally lower performing. Knowing nothing about our problem setting ahead of time, this might suggest one should conservatively prefer a robust objective to avoid potential catastrophic performance.

Analyzing the *projected* fixed-points. A major motivation of using the language of biconjugates to define the robust objectives is the ability to define our proxy objective without requiring access to the underlying state. In the last set of results, we did exactly this; we assumed access to the underlying state to define each objective, but artificially limited the agent to only use a limited set of features to estimate its value function. For this experiment, we remove this access to privileged information, defining each objective using only the set of features.

In Figure 6.4 we show the relative $\overline{\text{VE}^2}$ of each fixed-point to the best representable value function. The saturated colors (left-most blue and red bars) are the fixed-point without projection—that is, these are the same as shown in Figure 6.3. The unsaturated colors (right-most blue and red bars) are the fixed-point when both the primary and secondary variables use the same set of features. Finally, the intermediary colors (middle blue and red bars) are the fixed-point when the secondary variables are allowed a slightly larger function class, emulating an intermediate projection somewhere between the $\overline{\text{HBE}}$ and $\overline{\text{HPBE}}$ (and $\overline{\text{BE}^2}$ and $\overline{\text{PBE}^2}$ respectively).

In most cases, we find that the projected fixed-points provide better proxies for the $\overline{\text{VE}^2}$ than their non-projected counterparts. We also see empirically that the fixed-points of the $\overline{\text{HPBE}}$ and $\overline{\text{PBE}^2}$ are exactly equivalent in every setting, as we would have expected from Theorem 4. A natural conclusion, therefore, is that the primary role of the Huber function when optimizing these projected errors will be during the transient optimization, and not in the final asymptotic solution.

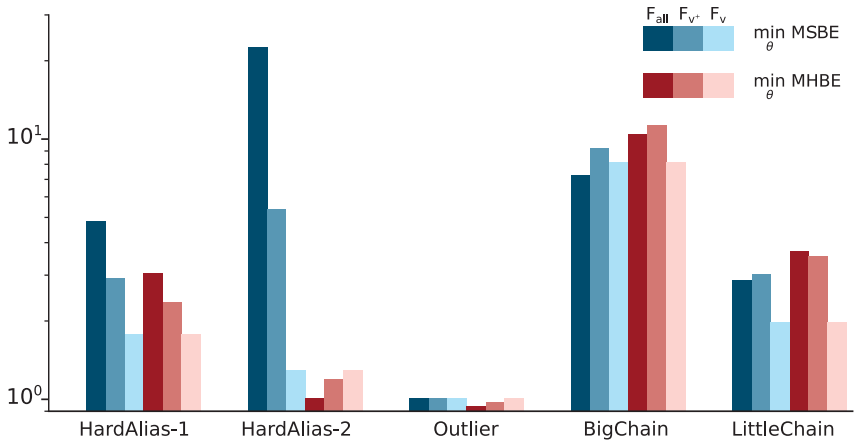

Figure 6.4: Evaluating the quality of fixed-points for the projected Bellman errors with three different projection sets. The more saturated colors (left) correspond to no projection; the less saturated colors (right) correspond to using $\mathcal{H} = \mathcal{V}$. The interim colors represent an intermediary projection which uses five additional features to fit the Bellman residual.

6.4 Algorithms for the Huber Bellman error

In this section, we derive gradient-based updates for the $\overline{\text{HPBE}}$ and $\overline{|\text{PBE}|}$. For a given h, the gradient with respect to w remains the same: $\sum_{s\in\mathcal{S}} d(s)h(s)\mathbb{E}_\pi\left[\nabla\delta(w)\,|\,S=s\right]$. This means that the job of selecting between the absolute, squared, and Huber Bellman errors rests solely on how we approximate the secondary variable, $h(s)$.

There are many ways to estimate h for the $\overline{\text{HPBE}}$ and $\overline{|\text{PBE}|}$. A natural starting point would be to use the same estimate, $\tilde{h}_\theta(s) \approx \mathbb{E}_\pi\left[\delta \mid S=s\right]$, as for the $\overline{\text{PBE}^2}$, then apply the corresponding non-linear function to $\tilde{h}$—sign or clipping. This gives the following updates for the objectives.

$$
\begin{aligned}
h(s_t) &= \text{sign}\left(\tilde{h}_{\theta_t}(s_t)\right) && \triangleright\ \overline{|\text{PBE}|}\\
h(s_t) &= \text{clip}_\tau\left(\tilde{h}_{\theta_t}(s_t)\right) && \triangleright\ \overline{\text{HPBE}}\\
h(s_t) &= \tilde{h}_{\theta_t}(s_t) && \triangleright\ \overline{\text{PBE}^2}
\end{aligned}
$$

$$\theta_{t+1} = \theta_t + \alpha_h\left(\delta_t - \tilde{h}_{\theta_t}(s_t)\right)\nabla_{\theta_t}\tilde{h}_{\theta_t}(s_t) \tag{6.3}$$

$$w_{t+1} = w_t + \alpha_v h(s_t)\left(\nabla_{w_t} v(s_t) - \gamma_{t+1}\nabla_{w_t} v(S_{t+1})\right) \tag{6.4}$$

Notice if we specifically parameterize $\tilde{h}_\theta(s) = \theta^\top x(s)$ and $v(s) = w^\top x(s)$, then we recover the GTD2 algorithm with linear function approximation [Sutton

et al., 2009]. Because the update for the primary weights is exactly the same as GTD2 and because the clip function encodes box-constraints on the secondary weights (and so is closed and convex), convergence of the GTD2-like algorithm for the $\overline{\text{HPBE}}$ follows directly from Nemirovski et al. [2009].

6.5 Empirical analysis in off-policy prediction

In this section, we empirically analyze the robust off-policy prediction algorithms in each diagnostic MDP.[1] Because we focus on the analytical optima in Section 6.3, and because each of the proposed algorithms are provably convergent, we focus instead on the transient optimization path for each algorithm. We seek to answer two primary research questions:

1. Are the robust optimization algorithms sample efficient compared to those that minimize the $\overline{\text{PBE}^2}$?

2. Are the robust algorithms insensitive to their stepsize parameter compared to the $\overline{\text{PBE}^2}$ minimizing algorithms?

In each of these problems, given sufficient data and an appropriate stepsize schedule, the learned value function should be identical to the analytical optima for the projected objectives found in Section 6.3. In fact, in Baird's counterexample problem, the optimal solution is identifiable making the asymptotic behavior of each optimization algorithm identical. Instead, we wish to understand the transient properties of these online learning algorithms. We will evaluate this both by investigating learning curves during early learning and also by using constant stepsizes, a common practice in life-long learning scenarios. To account for the effect of fixed stepsizes, we will investigate every algorithm at multiple levels of the stepsize parameter.

In Figure 6.5 we plot the $\overline{\text{VE}^2}$ averaged over time for each level of the stepsize parameter. Shaded regions show 95% percentile bootstrap confidence

[1] With the exception of the LongChain environment, which requires a very large number of samples to visit each endpoint regularly. Because of the high cost and because the fixed-point experiments suggest little difference in conclusions between SmallChain and BigChain, we choose to exclude this MDP.

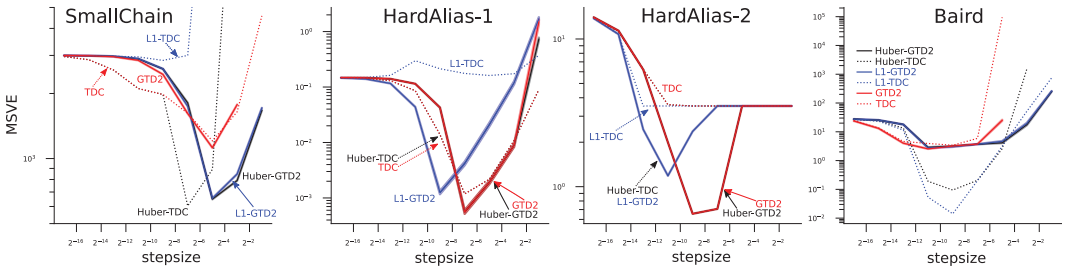

Figure 6.5: $\overline{\text{VE}^2}$ averaged over 100 independent trials for each stepsize in prediction domains. The mean squared algorithms generally performed well across environments—even the adversarially chosen environments—suggesting the difficulty in minimizing the $\overline{|\text{BE}|}$. The Huber algorithms performed best across many environments, often displaying less sensitivity to the choice of stepsize.

intervals using 100 experimental trials for every condition. Solid lines show the saddlepoint algorithms based on GTD2, which tend to perform far more conservatively than the dashed lines which show the gradient-correction formulation based on TDC. We also notice that algorithms which minimize the $\overline{|\text{PBE}|}$ typically perform much worse than either of the other objectives. This is not particularly surprising, as it is well-known that absolute losses are much more challenging to optimize than their squared counterparts.

6.6 Experiments in nonlinear control

In this section we empirically investigate the QRC-Huber algorithm. We first compare to several control algorithms in five classic control domains, investigating both learning speeds and stability in performance over different runs. We then show that QRC-Huber performs better *without* target networks, highlighting that these gradient algorithms may provide an alternative path to stabilizing DQN without slowing down learning. Finally, we conclude with a demonstration in a larger environment called Minatar.

6.6.1 Experiments in classic control environments

For the nonlinear control experiments, we investigate three classic control problems—Mountain Car, Cartpole, and Acrobot—from the Gym suite [Brockman et al., 2016], a larger domain with a heavily shaped reward, Lunar Lander,

and one additional domain designed to be particularly challenging for squared error algorithms, Cliff World. For all domains, we use discount factor $\gamma = 0.99$ and $\epsilon = 0.1$ for the ϵ-greedy policy. The episode is cut off if the agent fails to reach a terminal state in a pre-specified number of steps. When cut off, the agent is teleported back to the start state and does not update its value function, thus preventing the agent from bootstrapping over the teleportation transition.

We compare the QRC-Huber algorithm with three baseline algorithms from prior work. Because QRC-Huber builds on the QRC algorithm, we use this as a baseline to determine the impact of using a Huber Bellman error in place of a squared Bellman error. All design decisions and other elements of the update rule are the same between QRC and QRC-Huber. We additionally compare to SBEED [Dai et al., 2018], the original algorithm stemming from the conjugate Bellman error work. We modify the SBEED algorithm to use a direct mellowmax policy based on its estimated value function, instead of a parameterized policy as in its original implementation, in order to match the update rules of the other baselines. Finally, we compare to DQN [Mnih et al., 2013]. Due to DQN being a semi-gradient method, and because its use of a clipped error is only superficially a Huber error, we expect to find that DQN exhibits lower stability and higher sensitivity to hyperparameters.

For QRC-Huber, we fix the Huber threshold parameter $\tau = 1$ for all domains. For the QRC methods, we chose not to use target networks—a frozen, infrequently updated set of weights for the bootstrapping target—so that we can highlight the stability provided by using true gradient-based methods with robust losses. DQN and SBEED use targets networks and sweep over multiple refresh rates. DQN additionally sweeps over its Huber clipping parameter and SBEED sweeps over its objective tuning parameter η and mellowmax parameter λ. In total, QRC and QRC-Huber tune over 6 hyperparameter combinations while DQN tunes over 120 and SBEED tunes over 360.

To demonstrate the stability of each algorithm, we report the full distribution of the performance metric over 100 independent trials for the best stepsize on each domain. We use the average return achieved over the last 25% of steps

as our performance metric. We expect algorithms which exhibit stable performance to have a narrow, approximately normal distribution centered around higher return, whereas we expect algorithms which are unstable to have wide performance distributions or even multi-modal distributions. Standard learning curves are reported in Figure 6.7.

Figure 6.6 shows the performance distributions of each tested algorithm. QRC-Huber exhibits narrow and approximately normal performance distributions for every domain, suggesting the stability of the algorithm over random seeds. The QRC algorithm performs reasonably on the Acrobot and Cartpole domains, but performs quite poorly on the Cliff World domain. Because QRC is based on the mean squared Bellman error, the poor performance on Cliff World is exactly as expected, since this domain was chosen adversarially to highlight challenges with mean squared errors. While DQN is based on a clipped loss function that appears similar to the mean Huber Bellman error, it does not seem to enjoy the same stability as QRC-Huber, with average performance far worse than QRC-Huber on four of five domains due to high bimodality or long-tailed performance distributions. The learning curves in Figure 6.7 further highlight that QRC-Huber is the most robust of the three, across all five problem setting, either having comparable or notably better performance. The poor performance of DQN on Mountain Car and Cartpole is counterintuitive; given a sufficient number of learning steps and a large enough delay between target network updates, DQN can perform well in these environments. This results in a fundamental trade-off between stable performance and sample efficiency. The experiment in Figure 6.7 favors sample efficiency due to its short length.

6.6.2 Experiments in Minatar

Finally, we demonstrate that QRC-Huber can scale to larger domains using more complex neural network architectures. We use the Minatar suite of five miniaturized Atari games which retain much of the complexity of the full Atari games, while considerably reducing the computational requirements and cost [Young and Tian, 2019]. We use a convolutional neural network architecture

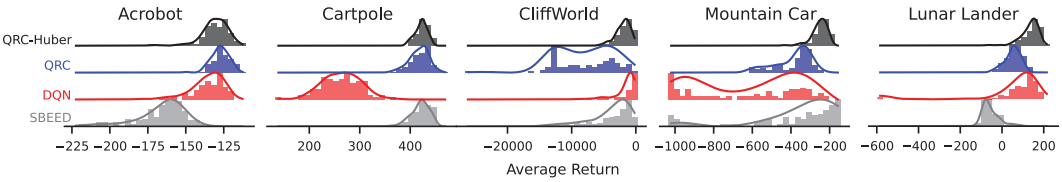

Figure 6.6: Performance distribution over 100 random seeds for the best hyperparameter setting chosen per-algorithm and per-domain. The performance measure is the average return over the last 25% of steps. QRC-Huber consistently has approximately normal and narrow distributions around high-performance returns. DQN has inconsistent behavior, with bimodal performance on Mountain Car and Lunar Lander, and long-tailed performance on Acrobot and Cartpole. SBEED has inconsistent performance with high-variance on several domains and long-tailed performance on CliffWorld and Mountain Car.

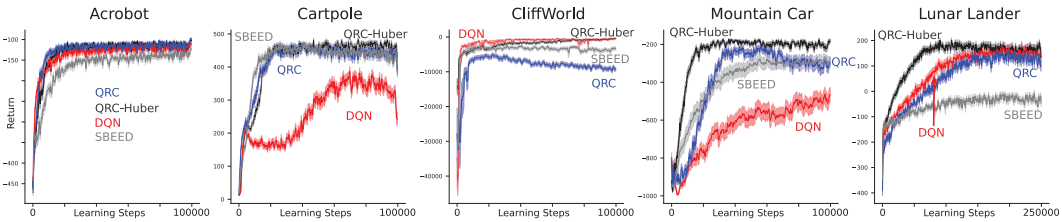

Figure 6.7: Learning curves for the best hyperparameter configuration for each domain, averaged over 100 random seeds. Shaded regions indicate one standard error. QRC-Huber is the only algorithm which is consistently among the best performing algorithms for every environment. DQN exhibits notable instability in both the Cartpole and Mountain Car environments, while QRC suffers from its squared loss in the adversarially designed CliffWorld environment. The SBEED algorithm consistently performs suboptimally on every domain except Cartpole, with notably worse performance on Lunar Lander.

with two hidden layers to learn value functions from images. These gradient-based learning rules without target networks, such as QRC and QRC-Huber, can outperform semi-gradient learning rules such as DQN, even when DQN is allowed to additionally tune its usage of target networks.

To avoid domain overfitting and reduce the cost of hyperparameter tuning, we treat the entire Minatar suite as a single problem setting. As such, each algorithm must pick one hyperparameter setting to use across all five games. The other benefit of this design is that it favors algorithms that are insensitive to hyperparameter choices. We allow all three control algorithms to sweep over

a small range of stepsizes and allow DQN to additionally sweep over target network refresh rates. We set the discount factor $\gamma = 0.99$ for all domains and otherwise use the same design parameters as Young and Tian [2019].

In order to compare performance of each agent, we average scores over each game using probabilistic performance profiles [Jordan et al., 2020, Barreto et al., 2010], then report the average scaled performance across the entire suite with 95% confidence intervals. We run each algorithm with its best hyperparameter setting for 30 runs on each game allowing comparisons on the Minatar suite using a total of 150 samples for each algorithm.

Table 6.1: Average performance on Minatar

QRC-Huber	0.53 ± 0.03
QRC	0.47 ± 0.02
DQN	0.36 ± 0.06

In Table 6.1, we report the average scaled return across games in the Minatar suite. Despite having four times the number of hyperparameter combinations and the ability to use target networks, DQN performs considerably worse than either gradient-based algorithm. Because QRC-Huber and QRC yield approximately normal performance distributions, we use a paired t-test and find that QRC-Huber has a statistically significant performance increase over QRC. DQN's performance profile is notably skewed by a small number of failing runs, however the difference in performance between DQN and either gradient-based algorithm is significant according to both a paired t-test and a much less powerful bootstrap t-test (which allows for a skewed sampling distribution). That QRC and QRC-Huber perform similarly is unsurprising as the largest possible reward in any Minatar game is +1, a design decision made in part because many algorithms—such as DQN—are unstable when learning from large rewards.

6.6.3 Omitting target networks

A primary motivation for building on gradient TD methods is that they are a theoretically sound way to obtain stable, convergent TD methods. Target net-

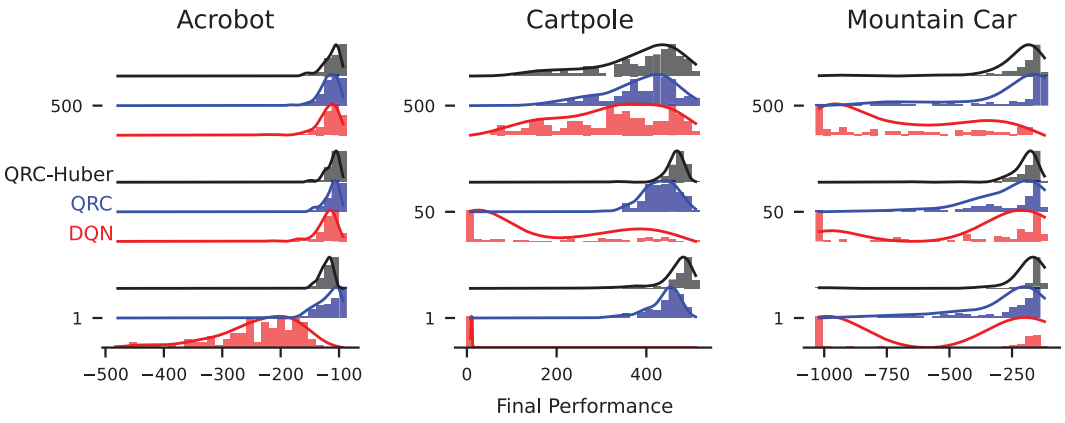

Figure 6.8: Ablating the impact of the target network refresh rate (1, 50 and 500) on the performance of the nonlinear control algorithms. A refresh rate of 1 means no target networks are used. DQN requires target networks to achieve above random performance on Cart-pole and to reduce the bimodality of its performance on Mountain Car. Even with target networks, DQN still exhibits large skew and bimodality in its performance distributions, indicating instability. The gradient methods QRC-Huber and QRC both perform better *without* target networks (the bottom row).

works currently lack theoretical justification, but are believed to empirically improve the stability of semi-gradient learning rules such as in DQN. Considering the non-negligible overlap in intended use case between gradient algorithms and target networks, we test for interactions between these algorithmic tools by extending the gradient algorithms to include target networks and measuring the change in performance. In this section, we address the question: "could gradient-based algorithms benefit from the use of target networks?".

We introduce target networks into QRC and QRC-Huber and sweep over the refresh rate—the number of learning steps before the target network weights are overwritten with the weights of the current network. If the target network refresh rate is one, then target networks are not used. We report the distribution of final performance over 100 random seeds, in Figure 6.8. The gradient-based algorithms perform better without target networks across environments. Because these algorithms already exhibit high stability, adding target networks serves only to harm the sample efficiency of the algorithms. DQN always required target networks in order to solve any of the tested problems, while requiring environment-specific tuning of the target network parameter.

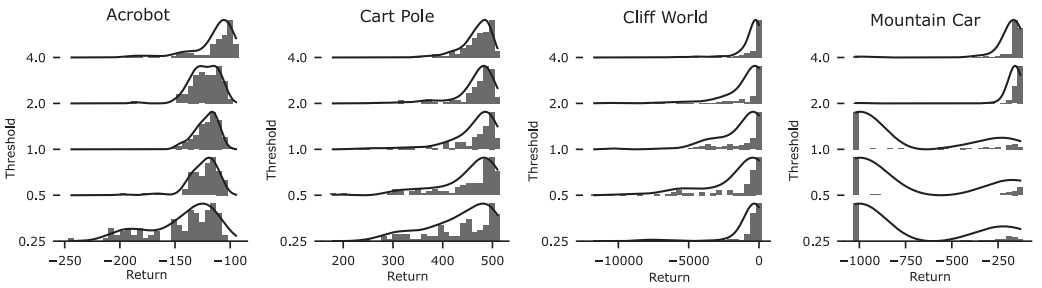

Figure 6.9: Ablating the impact of the threshold parameter for the Huber loss function for the QRC-Huber algorithm across the benchmark domains. For three of the domains, QRC-Huber is robust to the choice of threshold parameter with a default value of $\tau = 1$ being a good choice. However, the Mountain Car domain shows high bimodality in performance distribution across multiple random initializations of the neural network for smaller values of the threshold parameter.

On the excluded CliffWorld environment, we found that all algorithms performed best without target networks. This result is unsurprising given the simplicity of the environment dynamics: CliffWorld is designed to show differences in fixed-points, rather than differences during learning. We exclude Lunar Lander due to the high computational cost of performing extensive parameter sweeps and because QRC-Huber clearly already outperforms DQN on Lunar Lander *without* target networks as shown in Figure 6.6.

6.6.4 Ablating design decisions

In this section, we ablate the threshold parameter of the Huber function, τ, as well as the stepsize for each algorithm. While most of our domains we could reasonably pick a default value of $\tau = 1$ and avoid allowing QRC-Huber more opportunities to tune hyperparameters, this choice significantly impacted the performance of QRC-Huber on the Mountain Car domain. The choice of τ depends on the magnitude of the TD errors experienced by the algorithm during optimization—which is driven in-part by the magnitude of the rewards—and thus is domain-dependent. Similar to QRC, we could consider scaling the magnitude of rewards in a domain-independent way, for instance using the PopArt algorithm [Hessel et al., 2018b].

In Figure 6.9, we ablate over several choices of threshold parameter for the

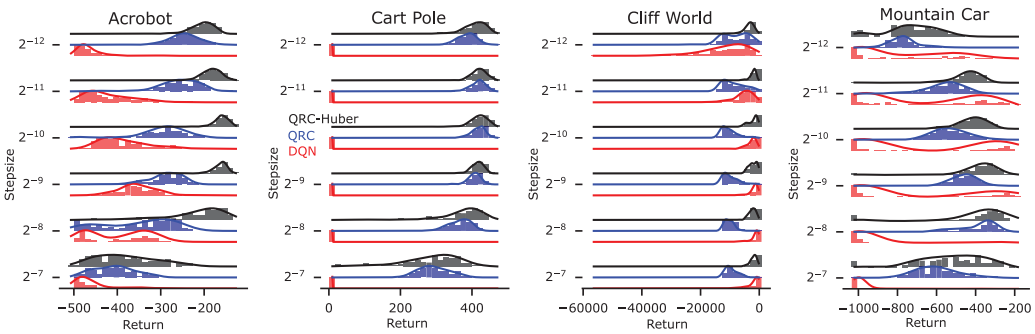

Figure 6.10: Comparing algorithms on benchmark control domains with the area under the learning curve as the performance metric. By including early learning in the performance metric, we get a sense of the sample complexity of each algorithm. QRC-Huber tends to perform favorably across all four domains compared to QRC and DQN, exhibiting much more narrow performance distributions that are often centered around higher rewards than the competitor algorithms.

QRC-Huber algorithm. We additionally investigate DQN's threshold parameter, however due to the general instability of DQN there was little pattern to the results, and so we omit them here for ease of visualization. For QRC-Huber, we see long-tail performance distributions as the threshold parameter is made smaller, likely due to the lower sample efficiency of the Huber algorithm when the optimization process is in the flat regions of the loss. On the Mountain Car domain, both QRC-Huber and DQN were significantly impacted by $\tau < 2$ and saw strongly bimodal performance distributions.

In Figure 6.10, we investigate the performance distribution of every tested stepsize for each algorithm. The poor performance of DQN—especially in the Cartpole domain—is surprising; likely stemming from DQN's incorrect use of the Huber loss function. A similar effect was noticed in Obando-Ceron and Castro [2021], where they ablated the choice of using a Huber loss versus a squared loss for DQN, finding that the squared loss consistently outperformed the Huber loss across several small control environments. Both of the gradient methods generally perform well across several values of their stepsize, with the most notable differences occurring in the Cliff World domain where QRC-Huber tends to significantly outperform QRC (note the wide scale of the horizontal axis caused by rare failing runs for DQN).

6.7 Summary

In this chapter, we introduced the biconjugate forms of the mean absolute Bellman error ($\overline{|\text{BE}|}$) and the mean Huber Bellman error ($\overline{\text{HBE}}$). Rewriting these errors in terms of their biconjugates allows us to optimize these objectives using gradient descent without suffering from the double sampling problem. This rewrite also allows us to constrain the scope of the objective to only evaluate the agent based on observed data, causing the biconjugate forms to be *learnable* in the sense described in Chapter 11 of Sutton and Barto [2018]. While these errors are not necessarily constructed from orthogonal projections, as in the case of the $\overline{\text{PBE}^2}$, the constrained inner optimization still allows identifiability.

We empirically evaluated the quality of the solutions for each proposed objective across several carefully designed diagnostic MDPs. These MDPs allowed us to highlight instances where robust losses define considerably better solutions than their squared counterparts, while also highlighting instances where a simple squared objective is preferable. We found that in all cases, when the solutions defined by the squared losses outperformed those by the robust losses, the difference in performance was marginal. Conversely, when the solutions of the robust losses performed better, the difference in performance was substantial.

We then derived online learning algorithms to minimize both the $\overline{|\text{BE}|}$ and $\overline{\text{HBE}}$ using stochastic samples. We built on the TDRC algorithm defined in Chapter 4, making light modifications to the secondary weights to control which loss was to be minimized. We empirically evaluated the performance of these algorithms on the same set of diagnostic MDPs, comparing to the performance of gradient TD methods that minimize squared losses. We found that the gradient TD algorithms that minimize Huber losses performed consistently well across all diagnostic MDPs, while algorithms that minimize the absolute loss often performed worse than either their Huber or squared counterparts.

Finally, we derived algorithms for off-policy control that minimize the $\overline{\text{HBE}}$, again building on the highly related QRC algorithm presented in Chapter 4.

We showed that in all tested environments, QRC-Huber performed comparatively or better than any of the squared counterparts QRC-Huber and QRC both exhibited greater reliability than the semi-gradient methods, and QRC-Huber displayed additional robustness over QRC in the adversarially designed CliffWorld environment.

The next and final, chapter of this thesis will summarize the insights from the prior few chapters. We will revisit the goal posed at the beginning of this thesis, and discuss how it has been addressed, the remaining gaps, and promising future directions.

Chapter 7

Conclusion

In this thesis, we have pursued a strategy to improve the reliability of off-policy reinforcement learning with nonlinear function approximation. We found that seemingly trivial modifications to the existing gradient TD methods lead to a surprising depth of consequences, ultimately enabling a sound, easy-to-use, and reliable gradient-based method for learning nonlinear value functions. We saw, empirically, that this method typically performed as well as unsound semi-gradient approaches, satisfying our goal of introducing reliable methods without sacrificing performance. We established that previous semi-gradient methods were unreliable in simple classic control environments, and that this lack of reliability ultimately harmed average performance. As a result, replacing these unreliable methods with gradient-based alternatives can ultimately lead to improved performance—on average—and an increase in trust for online reinforcement learning.

The first contribution of this thesis investigated which proxy objective function we should seek to minimize. This contribution lead to a generalization of the $\overline{\text{PBE}^2}$, the current most commonly used proxy objective in value function learning, and this generalization lead to several new perspectives on how to effective optimize the $\overline{\text{PBE}^2}$ with online, nonlinear algorithms.

Using the insights gleaned from the generalized $\overline{\text{PBE}^2}$, the second primary contribution of the thesis was the temporal difference learning with regularized corrections algorithm (TDRC). This algorithm takes advantage of the generalized projection set used in the biconjugate $\overline{\text{PBE}^2}$ reformulation of the prior

chapter, by imposing soft-constraints on the space of the secondary weights of the TDC algorithm. We showed theoretically that the TDRC algorithm provably stable under broad conditions, then we showed empirically that this stability translates to a significant improvement in reliability across several domains. We finally proposed a nonlinear control version of the TDRC algorithm, q-learning with regularized corrections, and again showed empirically that this provably stable algorithm leads to a significant improvement in reliability.

The third and final primary contribution of this thesis revisited the question of which objective function we should optimize. In this chapter, we further generalized the biconjugate Bellman objective to consider two convex statistically robust losses: the $\overline{\text{HBE}}$ and the $\overline{|\text{BE}|}$. We provided a detailed empirical investigation into the fixed-points of these losses, yielding insights into when statistical robustness leads to improved reliability in off-policy prediction problems. We then derived and empirically investigate algorithms that minimize these statistically robust losses, using the gradient-based methods studied in the previous chapter. We showed that these algorithms were empirically reliable, even in cases where gradient-based methods that minimize a squared loss were not.

7.1 Future research directions

Ultimately, this thesis carves a narrow path in addressing the issue of reliable reinforcement learning with nonlinear function approximation. Along this path, many open questions surfaced which we were unable to adequately address. In addition, there are many alternative paths we could have taken to reach this same conclusion. We will discuss a few such open questions and alternative paths that are promising directions for future work.

Emphatic state-weightings. In Chapter 3, we generalized the mean-squared projection Bellman error by considering different weightings over states used when averaging errors. This state distribution ultimately controls the tradeoff

of resources within a limitation function approximation class. In Figure 3.2, we showed the quality of the solutions for various proxy objective functions, measured by the squared distance of the objective's solution from the true value function. One such weighting that we considered was the emphatic weighting over states, $d_{\mathbf{m}}$ [Sutton et al., 2016, Hallak et al., 2016]. Surprisingly, we found empirically that objectives weighted by $d_{\mathbf{m}}$ typically defined better solutions than objectives weighted by either d_b or d_π. Unlike objectives weighted by d_π, there are multiple viable sample-based algorithms which minimize the $\overline{\text{PBE}^2}$ weighed by $d_{\mathbf{m}}$.

In considering potential paths to improve the reliability of reinforcement learning methods, one promising starting point was to build on emphatic TD methods. A primary motivation of emphatic TD is its provable convergence, even in the off-policy and function approximation setting [Sutton et al., 2016]—the same starting point provided by gradient TD methods. Unfortunately, unlike gradient TD methods, these theoretical stability guarantees of emphatic TD are not realized in conventional empirical settings, where emphatic TD's high-variance updates can cause divergence in domains like Baird's counterexample [Ghiassian et al., 2018, White and White, 2016]. Despite these challenges, the novel empirical insight provided by Chapter 3 heavily motivates revisiting emphatic TD methods to find novel strategies to reduce variance and incorporate emphatic TD with arbitrary nonlinear function approximation.

Controlling internal processes with error tracking. A primary insight of this work is the relationship between the secondary variables of the gradient TD methods and the expected temporal difference error. This work places heavy emphasis on devoting some of the agent's learning resources to tracking a statistic of its learning progress. In this work, we use this statistic to help control the direction of the semi-gradient updates and ultimately improve the reliability of the learning algorithm. There are, however, many potential uses for this internal statistic, and, if we are already devoting resources to learning it, we should use the estimated TD error as completely as possible. Here are

a few.

An increasingly common approach to improve the sample efficiency of deep reinforcement learning algorithms is to keep a priority ordering over data in the agent's replay memory [Schaul et al., 2016]. Similar to the choice of function class for the secondary weights, $\mathcal{H}$ (see Chapter 6), the prioritization of data in a replay buffer ultimately plays the role of reweighting the distribution of errors used to train the agent [Fujimoto et al., 2020]. This reweighting, however, cannot be faithfully realized by prioritized experience replay due to the extreme computational cost of maintaining accurate estimates of a sample's priority. This priority function also suffers from potentially high-variance from environmental sources, potentially allowing a non-trivial degree of reweighting based on noise which cannot be approximated by our function class; irreducible error.

Both of these challenges, however, can be addressed using an internal estimate of the expected TD error. By using a parametric estimate of the TD error, the irreducible component of the error is lost in the projection onto $\mathcal{H}$. This means the remaining component of the error is that which *can* be minimized. In addition, using an estimator which is cheap to evaluate—or at least cheaper to evaluate than the sampling cost of sampled TD errors—can alleviate the potential for staleness in the priority ordering. In the process of writing this thesis, we performed preliminary pilot investigations into the use of the secondary weights as a prioritization mechanism. Early results were encouraging and heavily suggested the utility in further pursuit of this direction of research.

Another clear opportunity is the use of the estimated TD error to form a control variate for the primary learning process, enabling lower variance learning targets for the value function update rule. Control variates are a popular form of variance reduction in the value function approximation literature [De Asis and Sutton, 2018, Jiang and Li, 2016, Thomas and Brunskill, 2016]. To form a control variate, we require a random variable—say δ—which shares some variation with the variable of interest—say the return G. We also require the variable used as a control variate, δ, to have zero mean, which we

could achieve through the replacement $Z = \delta - \mathbb{E}_\pi [\delta \mid x]$, where that conditional expectation can be approximated by $h(x) \approx \mathbb{E}_\pi [\delta \mid x]$. It is clear that δ and G share a large degree of covariance, particularly early in the learning process. It is unclear if the additional bias due to the approximation of $h(x)$ overcomes the variance reduction provided by using Z as a control variate.

A final potential utility of the error estimate $h(x)$ is for exploration. If the agent has an approximation of the amount of error it expects to receive throughout parts of the state-space, then, perhaps, we can use this error to drive the agent into parts of the environment where expects to receive the greatest learning signal. Such measures of learning potential have shown some promise as intrinsic rewards previously in the exploration literature [White et al., 2014, Linke et al., 2020].

The relationship between reliable learning algorithms and their hyperparameters. A primary challenge faced by this thesis, and really in most works in empirical reinforcement learning, is dealing with the ever-expanding space of hyperparameters. A central goal of this thesis was to improve the reliability of reinforcement learning algorithms, and a major component of reliability is having a smooth and predictable degradation of performance outside that algorithm's idealized configuration. In addition, a primary goal of improving the reliability of algorithms is to improve the reproducibility of scientific research. Unfortunately, large hyperparameter spaces and algorithms that are highly sensitive to their hyperparameters inhibit both of these goals.

Early in this thesis, we defined our problem setting as the continual learning problem where the agent interacts with its environment indefinitely. Our experiments, however, naturally had a finite time horizon after which they were terminated. The hyperparameters selected to evaluate our agents interact heavily with *when* we elected to terminate our experiments. As a concrete example, DQN can easily solve the Mountain Car classic control problem if given sufficient learning steps. DQN tends to favor very small stepsizes and very long target network refresh rates on this problem, both hyperparameter configurations harming DQN's sample efficiency. Had we decided to run

our experiments for millions of steps, this lower sample efficiency would have been dominated by the longer period of time after all methods had sufficiently learned, and differences between methods would have eventually vanished. Instead, by running Mountain Car for a small number of steps, we forced algorithms to select aggressive hyperparameters—large stepsizes and smaller target network refresh rates—which significantly altered our conclusions.

As a research community, we are only just starting to understand this relationship between hyperparameters, reliability, performance, and empirical design practices. There have been several recent works beginning to shed insight in this direction [Jordan et al., 2020, Eimer et al., 2023, Jayawardana et al., 2022, Benjamins et al., 2023], including our own work developed alongside this thesis [Patterson et al., 2023]. Yet the space of open questions in this area is innumerable. A particular direction of interest is understanding how hyperparameter configuration occurs in light of never-ending learning, where hyperparameter selection effectively becomes a zero-shot learning problem or, alternatively, must be completed entirely online through an agent's lifetime.

Adaptive capacity neural networks and the $\overline{\text{BE}}^2$. A final, and perhaps considerably more speculative, interesting direction opened up by the work in Chapter 3 as well as Schoknecht [2003], is the relationship between the $\overline{\text{BE}}^2$, the $\overline{\text{PBE}}^2$, and now the generalized $\overline{\text{PBE}}^2$ which can interpolate between these. Through the use of the triangle inequality, we can clearly articulate the error that is lost in the projection by the $\overline{\text{PBE}}^2$. Projecting away this error is a critical component of maintaining identifiability, which is important for value function learning. However, we can make use of this error as a mechanism for representation learning. This error describes the component of the value function which is not currently expressible with our features. Because the agent has control over its own features, for instance learning them through the use of a neural network, it has the ability to adapt its features in order to express a larger space of value functions. This adaptation may simply come from less aggressive filtration in earlier layers of the neural network, a process which is known to cause issues with never-ending learning [Lyle et al., 2022a,b,

Bellemare et al., 2019], or alternatively as a mechanism to inform the adaptive growth of capacity through adding new features to the function approximator.

7.2 Summary

This chapter completes the thesis by revisiting the primary objective of the thesis, and reiterating how each chapter partially addresses that objective. We conclude with several open research questions and alternative research paths highlighted by this thesis. We highlighted the potential utility of emphatic TD algorithms, which as yet are still under-explored for nonlinear function approximation. We pointed out the utility of tracking an internal estimate of a learner's error, and how this information can be used to inform many mechanisms of our complex learning systems. We lamented the challenges associated with massive-dimensional hyperparameter spaces and continual, online reinforcement learning and highlighted several promising directions. And finally, we speculated on the use of the information projected away by the $\overline{\text{PBE}^2}$ to inform representation learning, particularly with neural networks.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008. doi: 10.1007/s10994-007-5038-2.

Leemon Baird. Residual Algorithms: Reinforcement Learning with Function Approximation. *Machine Learning Proceedings*, pages 30–37, 1995. doi: 10.1016/B978-1-55860-377-6.50013-X.

Leemon C. Baird. *Reinforcement Learning through Gradient Descent*. PhD thesis, Brown University, 1999.

André M.S. Barreto, Heder S. Bernardino, and Helio J.C. Barbosa. Probabilistic performance profiles for the experimental evaluation of stochastic algorithms. *Conference on Genetic and Evolutionary Computation*, page 751, 2010. doi: 10.1145/1830483.1830617.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. doi: 10.1109/TSMC.1983.6313077.

Marc Bellemare, Will Dabney, Robert Dadashi, Adrien Ali Taiga, Pablo Samuel Castro, Nicolas Le Roux, Dale Schuurmans, Tor Lattimore, and Clare Lyle. A geometric perspective on optimal representations for reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

Marc G. Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C. Machado, Subhodeep Moitra, Sameera S. Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

Carolin Benjamins, Theresa Eimer, Frederik Schubert, Aditya Mohan, Sebastian Döhler, André Biedenkapp, Bodo Rosenhahn, Frank Hutter, and Marius Lindauer. Contextualize Me – The Case for Context in Reinforcement Learning, June 2023.

Shalabh Bhatnagar, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor–critic algorithms. *Automatica*, 45(11):2471–2482, November 2009. ISSN 00051098. doi: 10.1016/j.automatica.2009.07.008.

Alberto Bietti and Julien Mairal. Group invariance, stability to deformations, and complexity of deep convolutional representations. *The Journal of Machine Learning Research*, 20(1):876–924, 2019.

V. S. Borkar and S. P. Meyn. The O.D.E. Method for Convergence of Stochastic Approximation and Reinforcement Learning. *SIAM Journal on Control and Optimization*, 38(2):447–469, January 2000. ISSN 0363-0129. doi: 10.1137/S0363012997331639.

Justin A. Boyan. Technical Update: Least-Squares Temporal Difference Learning. *Machine Learning*, 49(2):233–246, November 2002. ISSN 1573-0565. doi: 10.1023/A:1017936530646.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. How Many Random Seeds? Statistical Power Analysis in Deep Reinforcement Learning Experiments. *arXiv:1806.08295*, July 2018.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms. *arXiv:1904.06979*, April 2019.

Bo Dai, Niao He, Yunpeng Pan, Byron Boots, and Le Song. Learning from Conditional Distributions via Dual Embeddings. *International Conference on Artificial Intelligence and Statistics*, page 10, 2017.

Bo Dai, Albert Shaw, Lihong Li, Lin Xiao, Niao He, Zhen Liu, Jianshu Chen, and Le Song. SBEED: Convergent Reinforcement Learning with Nonlinear Function Approximation. *International Conference on Machine Learning*, page 10, 2018.

Kristopher De Asis and Richard S. Sutton. Per-decision Multi-step Temporal Difference Learning with Control Variates. *Conference on Uncertainty in Artificial Intelligence*, July 2018.

Thomas Degris, Martha White, and Richard S. Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. OpenAI baselines, 2017.

Simon S. Du, Jianshu Chen, Lihong Li, Lin Xiao, and Dengyong Zhou. Stochastic Variance Reduction Methods for Policy Evaluation. *International Conference on Machine Learning*, 2017.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, page 39, 2011.

Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in Reinforcement Learning and How To Tune Them, June 2023.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Ma. Implementation Matters In Deep Policy Gradients: A Case Study On PPO and TRPO. *International Conference on Learning Representations*, page 14, 2019.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. *International Conference on Machine Learning*, 2018.

Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A Theoretical Analysis of Deep Q-learning. *Learning for Dynamics and Control*, pages 486–489, 2020.

Mattie Fellows, Matthew J. A. Smith, and Shimon Whiteson. Why Target Networks Stabilise Temporal Difference Methods. https://arxiv.org/abs/2302.12537v3, February 2023.

Werner Fenchel. On conjugate convex functions. *Canadian Journal of Mathematics*, 1(1):73–77, 1949.

Yihao Feng, Lihong Li, and Qiang Liu. A Kernel Loss for Solving the Bellman Equation. *Advances in Neural Information Processing Systems 32*, pages 15456–15467, 2019.

Scott Fujimoto, David Meger, and Doina Precup. An Equivalence between Loss Functions and Non-Uniform Sampling in Experience Replay. In *Advances in Neural Information Processing Systems*, volume 33, pages 14219–14230. Curran Associates, Inc., 2020.

Sina Ghiassian, Andrew Patterson, Martha White, Richard S. Sutton, and Adam White. Online Off-policy Prediction. *arXiv:1811.02597*, 2018.

Sina Ghiassian, Andrew Patterson, Shivam Garg, Dhawal Gupta, Adam White, and Martha White. Gradient Temporal-Difference Learning with Regularized Corrections. *International Conference on Machine Learning*, 2020.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

Leah Hackman. *Faster Gradient-TD Algorithms*. PhD thesis, University of Alberta, 2013.

Assaf Hallak, Aviv Tamar, Remi Munos, and Shie Mannor. Generalized Emphatic Temporal Difference Learning: Bias-Variance Analysis. *AAAI*, page 7, 2016.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning that Matters. *AAAI*, page 8, 2018.

J. Fernando Hernandez-Garcia and Richard S. Sutton. Understanding multi-step deep reinforcement learning: A systematic study of the DQN target. *arXiv preprint arXiv:1901.07510*, 2019.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018a.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task Deep Reinforcement Learning with PopArt. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(1), September 2018b.

Jiawei Huang and Nan Jiang. From Importance Sampling to Doubly Robust Policy Gradient. *International Conference on Machine Learning*, page 10, 2020.

Peter J. Huber. Robust estimation of a location parameter. In *Breakthroughs in Statistics*, pages 492–518. Springer, 1964.

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. *arXiv:1708.04133 [cs]*, August 2017.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement Learning with Unsupervised Auxiliary Tasks. *International Conference on Learning Representations*, 2016.

Vindula Jayawardana, Catherine Tang, Sirui Li, Dajiang Suo, and Cathy Wu. The impact of task underspecification in evaluating deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35: 23881–23893, 2022.

Nan Jiang and Lihong Li. Doubly Robust Off-policy Value Evaluation for Reinforcement Learning. *International Conference on Machine Learning*, May 2016.

Scott M. Jordan, Yash Chandak, Daniel Cohen, Mengxue Zhang, and Philip S. Thomas. Evaluating the Performance of Reinforcement Learning Algorithms. *International Conference on Machine Learning*, June 2020.

Seungchan Kim, Kavosh Asadi, Michael Littman, and George Konidaris. DeepMellow: Removing the Need for a Target Network in Deep Q-Learning. *International Joint Conference on Artificial Intelligence*, pages 2733–2739, August 2019. doi: 10.24963/ijcai.2019/379.

Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, page 15, 2015.

J Z Kolter. The Fixed Points of Off-Policy TD. *Advances in Neural Information Processing Systems*, page 9, 2011.

Cam Linke, Nadia M. Ady, Martha White, Thomas Degris, and Adam White. Adapting behavior via intrinsic reward: A survey and empirical study. *Journal of artificial intelligence research*, 69:1287–1332, 2020.

Michael L. Littman and Richard S Sutton. Predictive Representations of State. *Advances in Neural Information Processing Systems*, pages 1555–1561, 2002.

Bo Liu, Ji Liu, Mohammad Ghavamzadeh, Sridhar Mahadevan, and Marek Petrik. Proximal Gradient Temporal Difference Learning Algorithms. *International Joint Conference on Artificial Intelligence*, page 5, 2016.

Hao Liu, Yihao Feng, Yi Mao, Dengyong Zhou, Jian Peng, and Qiang Liu. Action-dependent Control Variates for Policy Optimization via Stein's Identity. *International Conference on Learning Representations*, February 2018.

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and Preventing Capacity Loss in Reinforcement Learning, May 2022a.

Clare Lyle, Mark Rowland, Will Dabney, Marta Kwiatkowska, and Yarin Gal. Learning dynamics and generalization in deep reinforcement learning. In *International Conference on Machine Learning*, pages 14560–14581. PMLR, 2022b.

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research*, 61:523–562, March 2018. ISSN 1076-9757. doi: 10.1613/jair.5699.

Hamid Reza Maei. *Gradient Temporal-Difference Learning Algorithms*. PhD thesis, University of Alberta, Edmonton, 2011.

Hamid Reza Maei, Csaba Szepesvári, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S Sutton. Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation. *Advances in Neural Information Processing Systems*, pages 1204–1212, 2009.

Sridhar Mahadevan, Bo Liu, Philip Thomas, Will Dabney, Steve Giguere, Nicholas Jacek, Ian Gemp, and Ji Liu. Proximal Reinforcement Learning: A New Theory of Sequential Decision Making in Primal-Dual Spaces. *arXiv:1405.6757*, 2014.

Ashique Rupam Mahmood, Huizhen Yu, and Richard S. Sutton. Multi-step Off-policy Learning Without Importance Sampling Ratios. *arXiv:1702.03006*, 2017.

Odalric-Ambrym Maillard, Remi Munos, Alessandro Lazaric, and Mohammad Ghavamzadeh. Finite-Sample Analysis of Bellman Residual Minimization. *Asian Conference on Machine Learning*, page 16, 2010.

Travis Mandel, Yun-En Liu, Sergey Levine, Emma Brunskill, and Zoran Popovic. Offline policy evaluation across representations with applications to educational games. In *AAMAS*, volume 1077, 2014.

Shie Mannor, Duncan Simester, Peng Sun, and John N. Tsitsiklis. Bias and variance approximation in value function estimates. *Management Science*, 53(2):308–322, 2007.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. page 9, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, and Georg Ostrovski. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Andrew William Moore. *Efficient Memory-Based Learning for Robot Control*. PhD thesis, University of Cambridge, 1990.

Jean Jacques Moreau. Inf-convolution, sous-additivité, convexité des fonctions numériques. *Journal de Mathématiques Pures et Appliquées*, 1970.

Remi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and Efficient Off-Policy Reinforcement Learning. *Advances in Neural Information Processing Systems*, page 9, 2016.

Prabhat Nagarajan, Garrett Warnell, and Peter Stone. Deterministic Implementations for Reproducibility in Deep Reinforcement Learning. *arXiv:1809.05676 [cs]*, June 2019.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

Johan S. Obando-Ceron and Pablo Samuel Castro. Revisiting Rainbow: Promoting more insightful and inclusive deep reinforcement learning research. *International Conference on Machine Learning*, 2021.

Cosmin Paduraru. Off-policy evaluation in Markov decision processes. 2013.

Andrew Patterson, Victor Liao, and Martha White. Robust losses for learning value functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–12, 2022a. ISSN 1939-3539. doi: 10.1109/TPAMI.2022.3213503.

Andrew Patterson, Adam White, and Martha White. A Generalized Projected Bellman Error for Off-policy Value Estimation in Reinforcement Learning. *Journal of Machine Learning Research*, 2022b.

Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical Design in Reinforcement Learning. *In Submission - Journal of Machine Learning Research*, 2023.

Alexandre Piché, Valentin Thomas, Rafael Pardinas, Joseph Marino, Gian Maria Marconi, Christopher Pal, and Mohammad Emtiyaz Khan. Bridging the Gap Between Target Networks and Functional Regularization, September 2023.

Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché-Buc, Emily Fox, and Hugo Larochelle. Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program). *arXiv:2003.12206*, April 2020.

Doina Precup, Richard S Sutton, and Satinder Singh. Eligibility Traces for Off-Policy Policy Evaluation. *International Conference on Machine Learning*, page 9, 2000.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 2021.

Touqir Sajed, Wesley Chung, and Martha White. High-confidence error estimates for learned value functions. *arXiv:1808.09127 [cs, stat]*, August 2018.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. *International Conference on Learning Representations*, February 2016.

Bruno Scherrer. Should one compute the Temporal Difference fix point or minimize the Bellman Residual? The unified oblique projection view. *International Conference on Machine Learning*, 2010.

Matthew Schlegel, Andrew Jacobsen, Zaheer Abbas, Andrew Patterson, Adam White, and Martha White. General value function networks. *Journal of Artificial Intelligence Research*, 70:497–543, 2021.

Ralf Schoknecht. Optimality of Reinforcement Learning Algorithms with Linear Function Approximation. *Advances in Neural Information Processing Systems*, page 8, 2003.

Max Schwarzer, Nitarshan Rajkumar, Michael Noukhovitch, Ankesh Anand, Laurent Charlin, R. Devon Hjelm, Philip Bachman, and Aaron C. Courville. Pretraining representations for data-efficient reinforcement learning. *Advances in Neural Information Processing Systems*, 34:12686–12699, 2021.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, pages 1038–1044, 1996.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, November 2018. ISBN 978-0-262-35270-3.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1.

Richard S. Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. *International Conference on Machine Learning*, pages 1–8, 2009. doi: 10.1145/1553374.1553501.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction. *International Conference on Autonomous Agents and Multi-Agent Systems*, page 8, 2011.

Richard S Sutton, A Rupam Mahmood, and Martha White. An Emphatic Approach to the Problem of Off-policy Temporal-Difference Learning. *Journal of Machine Learning Research*, page 29, 2016.

Brian Tanner and Richard S. Sutton. TD(λ) networks: Temporal-difference networks with eligibility traces. *International Conference on Machine Learning*, pages 888–895, 2005. doi: 10.1145/1102351.1102463.

Philip S Thomas and Emma Brunskill. Data-Efficient Off-Policy Policy Evaluation for Reinforcement Learning. *International Conference on Machine Learning*, page 10, 2016.

Ahmed Touati, Pierre-Luc Bacon, Doina Precup, and Pascal Vincent. Convergent Tree Backup and Retrace with Function Approximation. *International Conference on Machine Learning*, page 10, 2018.

J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42 (5):674–690, 1997. doi: 10.1109/9.580874.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep Reinforcement Learning and the Deadly Triad. *arXiv:1812.02648*, December 2018.

Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. *Advances in Neural Information Processing Systems*, pages 4287–4295, 2016.

Han Wang, Archit Sakhadeo, Adam White, James Bell, Vincent Liu, Xutong Zhao, Puer Liu, Tadashi Kozuno, Alona Fyshe, and Martha White. No More Pesky Hyperparameters: Offline Hyperparameter Tuning for RL, May 2022.

Marshall Wang, John Willes, Thomas Jiralerspong, and Matin Moezzi. A Comparison of Classical and Deep Reinforcement Learning Methods for HVAC Control, August 2023.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. *International Conference on Machine Learning*, page 9, 2016.

Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989.

Adam White. *Developing a Predictive Approach to Knowledge*. PhD thesis, University of Alberta, 2015.

Adam White and Martha White. Investigating practical linear temporal difference learning. *International Conference on Autonomous Agents and Multi-Agent Systems*, 2016.

Adam White, Joseph Modayil, and Richard S. Sutton. Surprise and curiosity for big data robotics. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Martha White. Unifying Task Specification in Reinforcement Learning. *International Conference on Machine Learning*, page 9, 2017.

Shimon Whiteson, Brian Tanner, Matthew E. Taylor, and Peter Stone. Protecting against evaluation overfitting in empirical reinforcement learning. *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 120–127, April 2011. ISSN 2325-1867. doi: 10.1109/ADPRL.2011.5967363.

Ronald J Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. 1992.

Tengyu Xu, Zhuoran Yang, Zhaoran Wang, and Yingbin Liang. Doubly Robust Off-Policy Actor-Critic: Convergence and Optimality. *arXiv preprint arXiv:2102.11866*, 2021.

Kenny Young and Tian Tian. MinAtar: An Atari-Inspired Testbed for Thorough and Reproducible Reinforcement Learning Experiments. *arXiv:1903.03176*, June 2019.

Shangtong Zhang, Hengshuai Yao, and Shimon Whiteson. Breaking the Deadly Triad with a Target Network. In *Proceedings of the 38th International Conference on Machine Learning*, pages 12621–12631. PMLR, July 2021.

Appendix A

Algorithms

For all algorithms, let

$$\begin{aligned}\rho_t &= \frac{\pi(A_t|s_t)}{b(A_t|s_t)}\\ \delta_t &= R_{t+1} + \gamma_{t+1} v(S_{t+1}) - v(s_t)\\ \theta_{t+1} &= \theta_t + \alpha_h \left(\rho_t \delta_t - \tilde{h}_{\theta_t}(s_t)\right) \nabla_{\theta_t} \tilde{h}_{\theta_t}(s_t)\end{aligned}$$

for fixed target policy π and behavior policy b.

GTD2

$$\begin{aligned}h(s_t) &= \tilde{h}_{\theta_t}(s_t)\\ w_{t+1} &= w_t + \alpha_v h(s_t) \left(\nabla_{w_t} v(s_t) - \rho_t \gamma_{t+1} \nabla_{w_t} v(S_{t+1})\right)\end{aligned}$$

GTD2-Huber

$$\begin{aligned}h(s_t) &= \text{clip}_\tau \left(\tilde{h}_{\theta_t}(s_t)\right)\\ w_{t+1} &= w_t + \alpha_v h(s_t) \left(\nabla_{w_t} v(s_t) - \rho_t \gamma_{t+1} \nabla_{w_t} v(S_{t+1})\right)\end{aligned}$$

GTD2-Abs

$$\begin{aligned}h(s_t) &= \text{sign} \left(\tilde{h}_{\theta_t}(s_t)\right)\\ w_{t+1} &= w_t + \alpha_v h(s_t) \left(\nabla_{w_t} v(s_t) - \rho_t \gamma_{t+1} \nabla_{w_t} v(S_{t+1})\right)\end{aligned}$$

TDC

$$\begin{aligned}h(s_t) &= \tilde{h}_{\theta_t}(s_t)\\ w_{t+1} &= w_t + \alpha_v \rho_t \left(\delta_t \nabla_{w_t} v(s_t) - \gamma_{t+1} h(s_t) \nabla_{w_t} v(S_{t+1})\right)\end{aligned}$$

TDC-Huber

$$\begin{aligned}h(s_t) &= \text{clip}_\tau \left(\tilde{h}_{\theta_t}(s_t)\right)\\ w_{t+1} &= w_t + \alpha_v \rho_t \left(\delta_t \nabla_{w_t} v(s_t) - \gamma_{t+1} h(s_t) \nabla_{w_t} v(S_{t+1})\right)\end{aligned}$$

TDC-Abs

$$h(s_t) = \text{sign}\left(\tilde{h}_{\theta_t}(s_t)\right)$$
$$w_{t+1} = w_t + \alpha_v \rho_t \left(\delta_t \nabla_{w_t} v(s_t) - \gamma_{t+1} h(s_t) \nabla_{w_t} v(S_{t+1})\right)$$

Appendix B

Proofs

B.1 Biconjugate proofs

Proposition 5 *The biconjugate of the square function* $f(x) = \frac{1}{2}x^2$ *is* $f^{**}(x) = \max_{h\in\mathbb{R}} hx - \frac{1}{2}h^2$.

Proof: Recall the definition of the convex conjugate and correspondingly the biconjugate:

$$f^*(x) = \sup_{h\in\mathbb{R}}\{hx - f(h)\}$$
$$f^{**}(x) = \sup_{h\in\mathbb{R}}\{hx - f^*(h)\}.$$

Then the conjugate of the square function with dual parameter a,

$$f^*(x) = \sup_{a\in\mathbb{R}} xa - \frac{1}{2}a^2.$$

Applying the convex conjugate again and we obtain

$$f^{**}(x) = \sup_{h\in\mathbb{R}}\left(xh - \sup_{a\in\mathbb{R}}\left(ha - \frac{1}{2}a^2\right)\right).$$

Clearly, the inner supremum is achieved at $a^* = h$, so plugging in the maximizing value of a, we obtain

$$\begin{aligned} f^{**}(x) &= \sup_{h\in\mathbb{R}}\left(xh - h^2 + \frac{1}{2}h^2\right) \\ &= \max_{h\in\mathbb{R}}\left(xh - \frac{1}{2}h^2\right). \end{aligned}$$

Finally multiplying by two, $2f(x) = x^2$ and $2f^{**}(x) = \max_{h\in\mathbb{R}}(2xh - h^2)$, arriving at the biconjugate of the square function used in Dai et al. [2017] and Dai et al. [2018]. □

Proposition 6 *The biconjugate of the absolute value function* $f(x) = |x|$ *is* $f^{**}(x) = \max_{h\in[-1,1]} xh$.

Proof: The proof follows the same format as the proof of the biconjugate for the square function. Defining the conjugate and biconjugate respectively,

$$f^*(x) = \sup_{a\in\mathbb{R}} xa - |a| = \begin{cases} 0 & \text{when } |x| \le 1 \\ \infty & \text{otherwise.} \end{cases}$$
$$f^{**}(x) = \sup_{h\in\mathbb{R}} xh - f^*(h).$$

Simplifying the biconjugate form, we get

$$\begin{aligned} f^{**}(x) &= \sup_{h\in\mathbb{R}} \begin{cases} xh & \text{when } |h| \le 1 \\ xh - \infty & \text{otherwise.} \end{cases} \\ &= \sup_{|h|\le 1} xh \quad \triangleright\ -\infty \text{ is not feasible} \\ &= \sup_{h\in[-1,1]} xh. \end{aligned}$$

Finally, considering the maximizing values of h, we get $h = -1$ when $x < 0$ and $h = 1$ with $x > 0$ so the biconjugate simplifies to

$$f^{**}(x) = \text{sign}\,(()\,x)x = |x| = f(x)$$

thus completing the proof. □

Proposition 7 *The biconjugate of the huber function is* $f_\tau^{**}(x) = \max_{h\in[-\tau,\tau]} xh - \frac{1}{2}h^2$.

Proof: Define the huber function as

$$p_\tau\,(\tau)\,a \overset{\text{def}}{=} \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \le \tau \\ \tau|a| - \frac{1}{2}\tau^2 & \text{otherwise.} \end{cases}$$

Then the convex conjugate is

$$f^*(x) = \sup_{a\in\mathbb{R}} xa - p_\tau\,(\tau)\,a = \sup_{a\in\mathbb{R}} \begin{cases} xa - \frac{1}{2}a^2 & \text{if } |a| \le \tau \\ xa - \tau|a| + \frac{1}{2}\tau^2 & \text{otherwise.} \end{cases}$$

In order to resolve the supremum we must consider four cases, (a) when $|a| \le \tau$ and $|x| \le \tau$, (b) when $|a| > \tau$ and $|x| \le \tau$, (c) when $|a| \le \tau$ and $|x| > \tau$, and finally (d) when $|a| > \tau$ and $|x| > \tau$. Starting with condition (a), we have

$$\sup_{|a|\le\tau} xa - \frac{1}{2}a^2 = xa^* - \frac{1}{2}a^{*2} = \frac{1}{2}x^2$$

because solving for this optimization gives $a^* = x$, which is a feasible solution because $|x| \le \tau$. For condition (b), we have two subcases. When $0 < x \le \tau$, we have $a = \tau$ because

$$\sup_{|a|>\tau} xa - \tau|a| - \frac{1}{2}\tau^2 = \sup_{a>\tau}(x-\tau)a - \frac{1}{2}\tau^2 = x\tau - \frac{3}{2}\tau^2$$

and when $-\tau \leq x < 0$, we have $a = -\tau$ because

$$\sup_{|a|>\tau} xa - \tau|a| - \frac{1}{2}\tau^2 = \sup_{a<-\tau} xa - \tau|a| - \frac{1}{2}\tau^2 = -x\tau - \frac{3}{2}\tau^2.$$

Because $-\tau \leq x < 0$ we have $-x\tau - \frac{3}{2}\tau^2 \leq \frac{1}{2}x^2$, so the supremum is actually obtained in condition (a). The symmetric argument holds for the first subcase of condition (b) as well. Finally, consider condition (c), then

$$\sup_{|a|>\tau} xa - \tau|a| - \frac{1}{2}\tau^2 = \sup_{a>\tau}(x-\tau)a - \frac{1}{2}\tau^2 = \infty$$

and symmetrically condition (d),

$$\sup_{|a|>\tau} xa - \tau|a| - \frac{1}{2}\tau^2 = \sup_{a<-\tau} xa - \tau|a| - \frac{1}{2}\tau^2 = \infty$$

and the supremum will be ∞ when $|x| > \tau$.
Thus, the conjugate function is

$$f^*(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \leq \tau \\ \infty & \text{otherwise.} \end{cases}$$

Using this to compute the biconjugate, we obtain

$$\begin{aligned} f^{**}(x) &= \sup_{h\in\mathbb{R}} hx - f^*(h) = \sup_{h\in\mathbb{R}} \begin{cases} hx - \frac{1}{2}h^2 & \text{if } |h| \leq \tau \\ hx - \infty & \text{otherwise} \end{cases} \\ &= \max_{h\in[-\tau,\tau]} hx - \frac{1}{2}h^2 \end{aligned}$$

where again the constraints on the maximization come from encoding the infeasibility of $|h| > \tau$. □