University of Alberta

Faster Gradient-TD Algorithms

by

Leah Hackman

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

Master of Science

Department of Computing Science

Abstract

Gradient-TD methods are a new family of learning algorithms that are stable and convergent under a wider range of conditions than previous reinforcement learning algorithms. In particular, gradient-TD algorithms enable off-policy problems—problems where the distribution of the data is different from the distribution the learner seeks to learn about—while using function approximation in a data-efficient on-line manner. Despite these positive features, previous empirical work, though limited, suggests that gradient-TD methods are slower than they could be. One example of this slowness is in on-policy problems, where gradient-TD methods have been shown to be slower than conventional-TD methods in some cases (Maei, 2011). In this thesis, we examine this slowness through on- and off-policy experiments and introduce several variations of existing gradient-TD algorithms in search of faster gradient-TD methods. We then introduce hybrid gradient-TD methods, a class of algorithms unique in their ability to use conventional-TD and gradient-TD learning updates when appropriate. We introduce three algorithms, two of which are hybrid gradient-TD methods and close with the first experimental results. In particular, we present promising results which indicate one of our new algorithms provides the benefits of a hybrid gradient-TD method while outperforming previous gradient-TD methods.

Acknowledgements

Perhaps before I acknowledge anyone, I should acknowledge the fact that the road to this thesis has been longer and more winding than I, or any of my fellow travellers could have predicted. And so I must thank everyone for staying the course with me through thick and thin (and mixed metaphors).

Thank you to my supervisor, Rich Sutton, who always saw the path even when I couldn't.

From the lab, I want to thank Hamid Maei for going in circles with me when things were going over my head. Thank you to Patrick Pilarski, for his endless enthusiasm and research-uncle ways. Thank you to Gabor Bartok, and Martha & Adam White for their humour and everyday kindness. Really, my time at the university has been an absurdity of riches with regards to all the wonderful people I have met and so I will leave the rest unnamed, but thank you to everyone who has made my time at the UofA so much better.

Thank you Mike & Shayna Bowling and Joel Koop for all the Tuesdays. Thank you to Michaela et al. for my sanity. Thank you to Anna Koop for her unwavering friendship, support, and knitting advice. Thank you to Dan, Quinn, Arabesque, Geoff, and Zak for being the lucky charms in my balanced breakfast. Thank you to Hailey Markowski and Jeffrey Woodcroft for what can only be described as everything. Thank you to Andrew Butcher for loving me through the good, the bad, and stargaze. And lastly, thank you to my family for a lifetime of unconditional love, support, and inside jokes.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Gradient Temporal-Difference (Gradient-TD) methods are a family of recently developed algorithms in Reinforcement Learning (RL). While seemingly simple in idea and execution, these algorithms enable us to solve a very difficult and long-studied sub-problem in the RL field. In RL, a decision-maker tries to learn how best to behave by observing the effects of its actions and changing its behaviour accordingly. Gradient-TD methods enable the decision-maker (or agent) to efficiently learn the value of a variety of different ways of behaving from one stream of experience, while still guaranteeing that the agent's estimate of the value is continually improving. More formally, this sub-problem is called the off-policy learning problem.

In an off-policy problem, a learning *agent*'s goal is to learn the value of behaving according to a specific *target policy*, but must do so using data generated by behaving according to a different *behaviour policy.* It has been a long-standing problem in RL to find an online algorithm which is provably convergent on off-policy learning problems while using function-approximation. Solutions to this problem have existed but have often suffered from practicality issues—be it that they are computationally too slow, require too much memory, or that they have infeasible requirements to ensure convergence (Bradtke & Barto 1996; Baird 1995; Precup, Sutton, & Dasgupta 2001). The gradient-TD family of algorithms are unique in that not only are they provably convergent on off-policy problems while using function approximation, but they have linear memory and computation-per-time-step requirements, making them extremely lightweight and applicable methods.

While gradient-TD algorithms boast many desirable properties, the limited experimental work using these methods thus far indicates that the trade off for the benefits of gradient-TD algorithms is a slower convergence rate, particularly compared to the performance of conventional-TD approaches on on-policy problems (problems where the target and behaviour policy are the same)(Maei 2011).

In this thesis, we take a simple Markov Decision Process (MDP) environment—a variation of the Random Walk problem with actions—and through a series of experiments, explore how gradient-TD algorithms compare to conventional-TD methods at solving the state-action prediction problem using linear function approximation. We start by looking at GQ, GQ2, and GQ-NEU (the latter two methods are natural action-valued extensions of the standard state-valued GTD2 and GTD algorithms), using Expected Sarsa as a conventional-TD algorithm for comparison.This is the first experimental work published using this state-action value methods. From there, we successively explore variations on these methods in order to find faster gradient-TD methods, with varying degrees of success.

Following this exploration of state-action value gradient-TD algorithms, we then introduce the idea of hybrid gradient-TD methods—algorithms which automatically use conventional-TD learning updates on on-policy problems and gradient-TD learning updates on off-policy problems. These algorithms solve the problem of gradient-TD methods being slower than conventional-TD methods on on-policy problems and show promise in providing faster convergence on off-policy problems. We then introduce three algorithms, AB, Hybrid-GQ, and TDGQ: the first two algorithms are briefly discussed in an appendix of Maei's thesis (2011), and the third is a brand new method, making its first appearance in this thesis. Both Hybrid-GQ and TDGQ are hybrid gradient-TD methods. We present the first published results for all three algorithms and end with exciting results which show that Hybrid-GQ successfully solves our on-policy performance issue while out-performing all our gradient-TD methods in our experiments.

Chapter 2 begins the thesis with an introduction to Reinforcement Learning, all the associated necessary vocabulary and notation, and to the off-policy problem and existing solutions. Chapter 3 then introduces gradient-TD methods, and in particular, gives derivations for the three base gradient-TD algorithms used predominantly in this thesis: GQ, GQ2, and GQ-NEU. Following these two background chapters, chapters 4 and 5 are experimental chapters, exploring the performance of our gradient-TD algorithms on on-policy and off-policy problems using our Random Walk with actions environment. In these chapters we also introduce several new algorithms and algorithmic variations in search for faster performing gradient-TD methods, with varying degrees of success. Chapter 6 introduces the concept of hybrid-TD algorithms and presents three new algorithms—AB, Hybrid-GQ and TDGQ. We then present results using these new hybrid gradient-TD algorithms in chapter 7 on the same off-policy problem as used in chapter 5.

Chapter 2

Reinforcement Learning and Off-Policy Learning

In this thesis, there are two background chapters: one to outline the fundamental problem our work stems from, and a second to introduce the existing solution algorithms we will be working with in this thesis. This chapter is the former, providing background information on the reinforcement learning (RL) learning problem and the off-policy learning problem. We begin with a brief introduction to reinforcement learning and the standard notation used throughout this thesis, and we then touch on several key topics from reinforcement learning that will be important in this thesis: we discuss the difference between prediction and control problems, introduce the basic ideas behind Temporal Difference learning, and briefly go into the use of function approximation for learning value functions. In the second half of this chapter we introduce off-policy learning, discuss why solving off-policy learning problems has been challenging, and briefly explore various existing off-policy learning methods.

2.1 The Reinforcement Learning Problem

Reinforcement learning (RL) is an area of machine learning that focuses on online learning algorithms for learning to select actions when interacting with a world in order to maximize a scalar signal. The learning system is called the *agent* and the world it interacts with is called the *environment*. The signal to be maximized, which can have positive and/or negative values, is called the *reward*. The interaction between the agent and the environment consists of a temporal sequence of *states*, *actions*, and *rewards*. At each discrete *time step* $t = 0, 1, 2, 3, \ldots$, the learning agent receives a state $s_t \in S$, takes an action $a_t \in A$, and as a result receives a reward $r_t \in \mathbb{R}$ and the next state $s_{t+1} \in S$. Experience is thus a sequence:

$\dots s_{t-1}, a_{t-1}, r_t, s_t, a_t, r_{t+1}, s_{t+1}, \dots$. Note that time is an inherent and important aspect of experience in the RL framework, which is in stark contrast to many off-line learning settings.

Typically when talking about an environment, there are two important things one needs to know to define the dynamics of the world: how the environment selects the next state, and how the environment generates the rewards. These two defining questions are expressed as the next state transition probability distribution, which gives the probability of being in a state s given an agent's history of experience ($P\left(s_{t+1} = s' \mid s_t, a_t, s_{t-1}, a_{t-1} =, \dots, s_0, a_0\right)$), and the expected next reward, which is the expectation of the next reward r_{t+1} given an agent's history of experience ($\mathbb{E}\left[r_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0\right]$).

In the case of reinforcement learning, one often assumes (and often requires when proving the convergence of algorithms) that the environment can be modelled as a Markov decision process (MDP). An MDP has one defining property:

$$P\left(s_{t+1}, r_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0\right) = P\left(s_{t+1}, r_{t+1} \mid s_t, a_t\right) \tag{2.1}$$

This property (the Markov property) asserts that the state s_t is sufficient information to make decisions about the future—all future state transitions and rewards in an environment are independent of any experience before time t. Accordingly, under the Markov property, the next-state transition probability distribution can be simplified so that it is only conditioned on the most recent state and action:

$$\mathcal{P}^a_{s,s'} = P\left(s_{t+1} = s' \mid s_t = s, a_t = a\right). \tag{2.2}$$

The expected next reward can be similarly simplified and notated as:

$$\mathcal{R}^a_{s,s'} = \mathbb{E}\left[r_{t+1} \mid s_t = s, s_{t+1} = s', a_t = a\right]. \tag{2.3}$$

Combining the next-state transition probabilities and the expected next reward with knowledge of how the agent is selecting actions completely expresses the expected stream of experience of an agent.

On a high level, a reinforcement learning problem is really two interrelated problems: the control problem and the prediction problem. The control problem is focussed on how an agent should behave. Ultimately, the agent's goal is to behave in a way that maximizes reward however, while learning, the agent may need to behave in an exploratory manner in order to learn what actions are best. Formally, we refer to an agent's behaviour as its *behaviour policy* and we express a policy as a conditional probability distribution of how likely an agent is to take an action when in a given state: $P(a \mid s)$. We refer to the behaviour policy as μ and use it in notation as follows: $\mu(a \mid s) = P(a \mid s, \mu)$. [1]

The prediction problem is about predicting the *expected return* of the states—or state-action pairs—of an environment. In this thesis we exclusively focus on learning the expected return for state-action pairs and so our discussion will focus on this case. The return of a state-action pair, $< s, a >$, is the sum of all rewards experienced after taking action a leaving state s at time-step t:

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T,$$

This definition of the return makes sense when we consider problems that have a fixed start and end point (and thus we have a final reward r_T). Such problems are referred to as episodic and the return is the sum of the rewards experienced until the end of the episode. In the case where a problem has no end point we call the problem continuous and in this case we must use a discounted return,

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\inf} \gamma^k r_{t+k+1}. \tag{2.4}$$

The discounted return is also used in episodic problems with long episodes. From this point on, we use the discounted return in all equations as it is the more general case (note that when using a value of $\gamma = 1$, the discounted and non-discounted return are one and the same).

Note that the return is dependent upon the actions the agent selects. As such, the expected return of a state is dependent on a policy. We refer to the policy used in the prediction problem as the *target policy* and refer to this policy as π. The expected return of a state-action pair $< s, a >$ can then be written as $\mathbb{E}[R_t \mid s_t = s, a_t = a, \pi]$;

[1] Note that using μ for the behaviour policy goes against traditional notational convention: typically policies are abbreviated using the symbol π, where multiple policies are distinguished using subscripts. Here, because of the importance of distinguishing between the behaviour and target policy in off-policy problems, the symbol μ is used for the behaviour policy and π is reserved for the target policy. We ask readers to suspend their curiousity about target policies and off-policy problems for a moment longer and read on.

here t is any time-step and the expectation assumes that actions after time-step t are selected according to policy π. In an on-policy problem, the behaviour policy μ and target policy π are the same—the agent gathers data according to the policy μ, and simultaneously learns to predict the expected return for that same policy. This thesis is particularly concerned with off-policy problems—problems where the behaviour and target policy are different—and we will discuss this case in greater detail later in this chapter.

Returning again to our discussion of the prediction problem, it is the goal of the prediction problem to learn the expected return for all possible state-action pairs. We formulate this problem as needing to learn a *value function*, Q^π, which maps state-action pairs to their expected return: $Q^\pi : S \times A \to \mathbb{R}$. Accordingly, we often refer to the expected return of a state-action pair as the *value* of the pair. The true value function for the target policy π is defined as satisfying the following equation:

$$Q^\pi(s,a) = \mathbb{E}\left[R_t \mid s_t = s, a_t = a\pi\right] = \mathbb{E}\left[\sum_{i=t+1} \gamma^{i-1} r_i \mid s_t = s, a_t = a, \pi\right] \tag{2.5}$$

In this thesis, we denote the true target-policy value function as Q^π and denote learned approximations of the value function as $\hat{Q}$

In the full RL-problem, where we solve both the control and prediction problem at once, these two problems are inherently interrelated—the value function informs the policy about how valuable actions can be and, in turn, the policy is followed to gather data in order to learn the value function. In practice, an agent interleaves two learning steps: one step improving its value function and another step improving its policy. This cycle is repeated, with changes in one component affecting the other and vice versa, until convergence. When evaluating RL methods however, it is sometimes clearer and more informative to look at the agent's performance on just the control or prediction problem. In the case of this thesis, we will only be looking at our algorithms performance on the prediction problem. This simplification allows for easier analysis of results and is also a necessity given some of the algorithms presented in this thesis have not yet been proven to converge in the control case.

Before we can discuss gradient-TD methods in the next chapter, we pause here to explore some of the basic foundational methods for solving prediction problems. One approach to learning the value function, is to collect actual samples of the return and use an average of the returns experienced to learn the value for each state-action pair. This idea of averaging actual samples of the return is the Monte Carlo approach to solving the prediction problem (Sutton & Barto 1998). A learning agent using a Monte Carlo approach thus waits until a full return is sampled—at the end of one episode, or when enough experience has passed that γ^{t+k} is sufficiently close to zero— and then updates its estimate of the value for all states visited in that episode. An example of a simple Monte Carlo value function update is as follows:

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha \left[R_t - \hat{Q}(s_t, a_t) \right]. \tag{2.6}$$

Here, the value of the state action pair $< s, a >$ is improved by using the difference between the experienced return and the current estimate of the value. Monte Carlo methods have the benefit of needing no knowledge of the dynamics of the environment, but have the disadvantage of needing to wait until a full return is experienced before any learning can be done. This can be particularly problematic with long episodes or continuous learning problems.

An alternative approach to solving the prediction problem is *Temporal Difference* (TD) learning (Sutton 1984). TD-learning is one of the seminal reinforcement learning prediction methods. The core ideas of TD-learning are at the heart of many reinforcement learning methods, including the gradient-TD methods we will be looking at in this thesis. To understand the fundamental idea behind TD-learning we must first discuss the Bellman Equation[2] (Sutton & Barto 1998):

$$Q^{\pi}(s, a) = \sum_{s'} \mathcal{P}^{a}_{s,s'} \left[\mathcal{R}^{a}_{s,s'} + \gamma \sum_{a'} \pi(a' \mid s') Q^{\pi}(s', a') \right] \tag{2.7}$$

where s' is selected from the set of all possible next states given the agent is in state s and has taken action a, and a' is selected from the set of all actions possible in state s'. This equation gives a recursive relationship between the value of a state-action pair and the value of the next possible state-action pairs. [3]

If the environment is an MDP and one has the underlying next-state transition dis-

[2]We are discussing the Bellman Equation for state-action values. Typical discussions of the Bellman Equation will be phrased in terms of learning state values. For brevity and simplicity, we will simply refer to this as the Bellman Equation, and refrain from restating that this is the Bellman Equation for state-action values.

[3]To see why this recursive relationship works, consider that $Q(s, a) = \mathbb{E}[R_t \mid s_t = s, a_t = a, \pi]$. We can expand the expected return and rewrite this equation as: $Q(s, a) = \mathbb{E}[r_t + \gamma R_{t+1} \mid s_t = s, a_t = a\pi]$. From here, note that we can substitute our value function in for R_{t+1}, arriving at $Q(s, a) = \mathbb{E}[r_t + \gamma Q(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a\pi]$.

tribution, $\mathcal{P}^a_{s,s'}$, and expected next-reward, $\mathcal{R}^a_{s,s'}$, one could use dynamic programming to solve the Bellman equation. Dynamic programming is often not a feasible solution however, due to large state/action spaces and/or a lack of knowledge of the underlying MDP for an environment. Regardless, the Bellman equation gives us the pivotal idea of *bootstrapping*: using the relationship between neighbouring states in order to learn the value of one state-action pair using the value of neighbouring pairs.

One other important idea that we get from the Bellman equation is the *Bellman Error*. The Bellman error is the difference between the estimate of $Q^\pi(s,a)$ and the expected next reward and next state-action value, $(\mathcal{R}^a_{s,s'} + \gamma Q^\pi(s',a'))$. The Bellman error measures how inconsistent the value function is: is the estimate of the value for any state-action pair appropriate given the value estimates of its neighbours?

The concept behind TD-learning is to combine the idea of bootstrapping and minimizing the bellman error with the idea of using sampled rewards and state-action transitions from Monte Carlo methods. For the purposes of this thesis, we will discuss the Sarsa algorithm, the state-action compliment to the basic TD-learning algorithm (Rummery & Niranjan, 1994), to explain the fundamentals of TD-learning. At every time-step, Sarsa takes the current sampled transition $< s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1} >$ and updates its estimate $\hat{Q}(s_t,a_t)$ so that $\hat{Q}(s_t,a_t)$ and $\hat{Q}(s_{t+1},a_{t+1})$ are more consistent with one another:

$$\hat{Q}(s_t,a_t) \leftarrow \hat{Q}(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \hat{Q}(s_{t+1},a_{t+1}) - \hat{Q}(s_t,a_t) \right] \tag{2.8}$$

The estimate $\hat{Q}(s_t,a_t)$ is moved by a small amount (dictated by the size of the step-size parameter α) in the direction of a sample of the Bellman error. We refer to this sample of the Bellman error as the *TD-error* and often notate it as $\delta_t = r_{t+1} + \gamma \hat{Q}(s_{t+1}, a_t + 1) - \hat{Q}(s_t,a_t)$. Using δ in our notation, we can rewrite the Sarsa update as:

$$\hat{Q}(s_t,a_t) \leftarrow \hat{Q}(s_t,a_t) + \alpha\delta_t \tag{2.9}$$

Sarsa is thus able to use samples, like Monte Carlo methods, but avoids having to wait for the return of an episode. This means that Sarsa is able to update it's value function after ever step—learning is immediate. The use of the TD-error to gain the benefit of both sampling and bootstrapping is a very pervasive idea in the field of RL and many of the algorithms in the field draw their roots from this simple principle, including the gradient-TD algorithms we will look at in this thesis.

2.2 Sarsa and Expected Sarsa with Function Approximation

Until this point, our discussion of value functions has implied that one learns a unique value for each individual state-action pair. While a value function *can* be a table of learned values, with a value learned for every state-action pair, the state and action space of an environment is often too large for a table to be feasible. Value tables become too large and a large state-action space requires a lot of experience before an agent experiences every possible state-action pair at least once. Also, a table of values does not lend itself easily to generalization between similar states and actions. As such, some form of function approximation is often used.

When using function approximation, the learning problem includes a feature function, notated as $\phi(s, a)$, which maps states or state-action pairs to a vector of features. There are no constraints on what these features may be—they may be anything from binary values to labels—so long as they provide compatible input for the chosen function approximation technique. In the case of linear function approximation, which we focus on in this thesis, the agent learns a set of linear weights θ so that the value of a state or state-action pair is approximated by the dot product of θ and $\phi(s, a)$: $\hat{Q}(s, a) = \theta^\top \phi(s, a)$. In this thesis, we will be using linear function approximation for learning the value function in all of our learning algorithms.

Using linear function approximation, our TD-error becomes $\delta_t = r_{t+1} + \gamma\theta_t^\top \phi(s_{t+1}, a_{t+1}) - \theta_t^\top \phi(s_t, a_t)$ (Sutton & Barto 1998). Our Sarsa update then becomes:

$$\theta_{t+1} \leftarrow \theta_t + \alpha\delta_t\phi(s_t, a_t). \tag{2.10}$$

This should look similar to our update for Sarsa without function approximation, however, note that we multiply the TD-error by the current feature vector. In this way, the weights in θ are updated proportionally to how influential they were in generating the current TD-error: if a feature $\phi_i(s_t, a_t)$ was 0 for an update, the i-th learned weight (θ_i) did not (and cannot) contribute to the value of the current state-action pair, and thus that weight is not responsible for the TD-error generated by that transition and should not be updated. Similarly, if $\phi_i(s_t, a_t)$ was non-zero, then θ_i is able to impact the value of the current state-action pair, and thus that weight should be adjusted according to the TD-error.

In this thesis, part of our evaluation of gradient-TD methods involves comparing them to conventional-TD methods for on-policy problems. While Sarsa is truly a foundational conventional-TD method, we also will be looking at a Sarsa variant named Expected Sarsa (van Seijen, et al. 2009). We will address why Expected Sarsa is appropriate for our experiments in our on-policy experiment chapter (chapter 4). For now, we will only focus on presenting the algorithm.

While Sarsa takes it's name from the full sampled transition $< s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1} >$ used in its update, Expected Sarsa does not use a full sampled transition. In place of the experienced next action a_{t+1}, Expected Sarsa uses an expectation over the next action instead:

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \sum_a \pi(s_{t+1}, a)\hat{Q}(s_{t+1}, a) - \hat{Q}(s_t, a_t) \right]. \tag{2.11}$$

To simplify the update for Expected Sarsa with linear function approximation, we introduce a new piece of notation. When we use linear function approximation, the expectation term in our update $\sum_a \pi(s_{t+1}, a)\hat{Q}(s_{t+1}, a)$ is replaced with $\sum_a \pi(s_{t+1}, a)\theta^\top \phi(s_{t+1}, a)$. To simplify this, we introduce the notation[4] $\bar{\phi}^\pi(s) = \sum_a \pi(s, a)\phi(s, a)$, allowing us to rewrite the expected next state value as $\theta^\top \bar{\phi}^\pi(s_{t+1})$. Using this new notation, the update for Expected Sarsa with linear function approximation is

$$\theta_{t+1} \leftarrow \theta_t + \alpha \left[r_{t+1} + \gamma \theta_t^\top \bar{\phi}^\pi(s_{t+1}) - \theta_t^\top \phi(s_t, a_t) \right] \phi(s_t, a_t). \tag{2.12}$$

Using this expectation over the next action, Expected Sarsa has been shown to help reduce variance in updates, meaning larger values for the step size (α) are feasible in practice (van Seijen et al. 2009). This can speed up learning.

Up until this point, we have refrained from discussing one key element of TD methods: the *TD fixed-point*. Sometimes referred to as the TD-solution, or the LSTD-

[4]We will continue to use this notation throughout the thesis as it is also used by our gradient-TD methods. Here, we use the super script π to indicate that we're taking the expectation according to the target policy. Later, we will also use the superscript μ when discussing the expected state-action feature vector using the behaviour policy μ.

solution, the TD fixed-point is perhaps most appropriately named the linear-TD fixed-point as it is the fixed point of TD-learning when using linear function approximation. The TD fixed point occurs when learning effectively stops and the expected update, $\mathbb{E}\left[\delta_t \phi(s_t, a_t)\right]$ is zero. As such, the TD fixed-point, θ^*, is the point which satisfies:

$$0 = \mathbb{E}\left[(r_{t+1} + \gamma {\theta^*}^\top \phi(s_{t+1}, a_{t+1}) - {\theta^*}^\top \phi(s_t, a_t))\phi(s_t, a_t)\right] = \mathbb{E}\left[\delta_t \phi(s_t, a_t)\right] = -A\theta^* + b$$

where $A = \mathbb{E}\left[\phi(s_t, a_t)(\phi(s_t, a_t) - \gamma \phi(s_{t+1}, a_{t+1})^\top\right]$ and $b = \mathbb{E}\left[r_{t+1}\phi(s_t, a_t)\right]$.

Thus solving for θ^* gives us that $\theta^* = A^{-1}b$. Both Sarsa and Expected Sarsa converge to this fixed point, as do our gradient-TD methods which we will see soon.

2.3 Off-Policy Problems in Reinforcement Learning

An off-policy learning problem is one where the agent solves the prediction problem using one policy—the target policy π—while the agent is controlled by a different behaviour policy μ. Fundamentally, off-policy problems are a subclass of the broader learning problem: trying to learn to predict data drawn according to one distribution using data drawn from a different distribution. Expressed as this more general idea, this problem is not unique to reinforcement learning, however, we limit the breadth of our discussion here to off-policy problems in the context of reinforcement learning.

There can be many practical reasons why one might want to use different behaviour and target policies. One common reason for separating the target and behaviour policy is so that the agent can learn the value function for an optimal policy while using an exploratory policy to gather data. Q-learning is a well known example of an off-policy algorithm whose target policy is exploitative—using its current knowledge of the world to pick the best actions—while allowing for a much more exploratory behaviour policy (Watkins 1989). Another reason one may need to use different target and behaviour policies is when data may be difficult, expensive or dangerous to obtain. For example, one may want to learn the value of using a new control policy for monitoring a large factory: trying the new policy directly in order to learn whether the new policy is safe or profitable is somewhat counterproductive. It would be preferable if one could learn the value of the new policy using readily available existing data of the factory. Similarly, one may want to initially use data generated by a human controller for a robot so that the human can make the robot explore large parts of the space while avoiding damage to the robot. Lastly, off-policy learning is useful for those interested in learning multiple things at once. An agent may want to learn value functions for many possible target policies all at the same time from the same behaviour stream.

To understand the difficulty in solving off-policy problems let us reconsider the Bellman equation (equation 2.7) and how the Bellman equation factors into the update for Sarsa. In the Bellman equation we need to know $\mathcal{P}^a_{s,s'}$ and $\mathcal{R}^a_{s,s'}$ in order to relate $Q^\pi(s',a')$ to $Q^\pi(s,a)$. The Sarsa update relies on the fact that the probability distribution of the transition from s to s' is inherent in the frequency of these transitions occurring: transitions from s to s' will occur in our data with a frequency proportional to the true dynamics of the problem. Therefore, if we are twice as likely to transition to s'' as s' from state s, then we will see twice as many samples transitioning to s'', and we will update $\hat{Q}$ twice as often using s'', thus the value function incorporates the state transition model inherently. Herein lies the problem: our data is inherently selected according to the distribution μ but we want to learn the value $Q^\pi(s,a)$ according to the distribution π—thus we can no longer rely on the proportional frequency of states and actions in our data to provide an unbiased estimate of Q^π. Algorithms that try to solve this problem tend to fall into one of two camps: methods that remove the bias and methods that reframe the learning update.

The camp that wants to remove the bias is populated by methods that use the data as given but corrects for the bias so that the end result is an unbiased estimate of the value function according to π. Importance sampling techniques are an example of methods that remove the bias. In importance sampling methods, all updates are weighted according to a ratio of the likelihood of seeing the history of experience up to the current state when following the target policy versus following the behaviour policy (Precup, Sutton, & Dasgupta 2001). Unfortunately, importance sampling suffers from the fact that as histories become longer, this ratio has incredibly high variance which can cause computational issues where the ratio is too large or too small to be accurately represented.

The alternative approach is to reframe the problem: instead of learning the value for a state given the agent is always following policy π, we consider the problem where the agent has been behaving according to policy μ up until the current state and will henceforth follow policy π. We refer to this case as the *excursion* approach. Algorithms like Q-learning fall into this camp. When updating the value function using a single transition, these methods accept that s or $< s, a >$ are drawn according to μ and thus use the sample from the true data stream. Excursion methods then select a different s' or $< s', a' >$ that follows the target policy and use this half-real-and-half-imaged transition to update their value function. Such excursion methods still learn biased value functions, however the fact that they do not suffer from the computation issues that plague bias free methods makes the trade-off seem favourable. This thesis exclusively examines excursion methods for off-policy learning.

A potential problem for TD-based excursion methods arrises however when function approximation is used. Unfortunately, while TD learning has been shown to be convergent on off-policy problems when using a table look up value function, counterexamples exist to show that TD can diverge when using function approximation on off-policy problems. One such counterexample is Baird's Off-policy Counterexample (Baird 1995; Sutton & Barto 1998). Figure 2.1 shows the 7-state MDP used in Baird's counterexample. The reward in this MDP is zero on all transitions, meaning the true value function for any policy should be zero for all states. The target policy for this problem selects the solid line actions with a probability of 1. The behaviour policy selects the solid line actions with a probability of 1/7, and the dotted line actions with a probability of 6/7. All actions are deterministic, meaning selecting the action to move from s_i to s_j is always successful.

The value function is approximated by a set of linear weights θ with 8 weights, one more than the number of states in the MDP. For states $\{s_1, \dots, s_6\}$, the value of a state i (where V notates the state value function, analogous to the use of Q^π for the state-action value function) is $V(s_i) = 2\theta(i) + \theta(0)$ and the value of the last state s_7 is $V(s_7) = \theta(7) + 2\theta(0)$. The discount factor is $\gamma = 0.99$. The solution for this problem is simple—$\theta(i) = 0, i \in \{0, \dots, 7\}$—however, Baird has shown that both TD(0) and dynamic programming with incremental updates (TD's approximate dynamic programming counterpart) will diverge when trying to solve this problem.

Until recently, almost all existing TD-based excursion methods that have been proven to remain convergent when using function approximation on off-policy problems have suffered from potentially problematic algorithmic or convergence requirements. The Least Squares Temporal Difference (LSTD) method is one method that solves for the TD solution and has been shown to be convergent on off-policy problems using linear function approximation, however LSTD requires $\mathcal{O}\left(n^2\right)$ memory and per-time-step computation, where n is the number of features used (Bradtke & Barto 1996). The incremental version iLSTD, improves upon this, reducing the

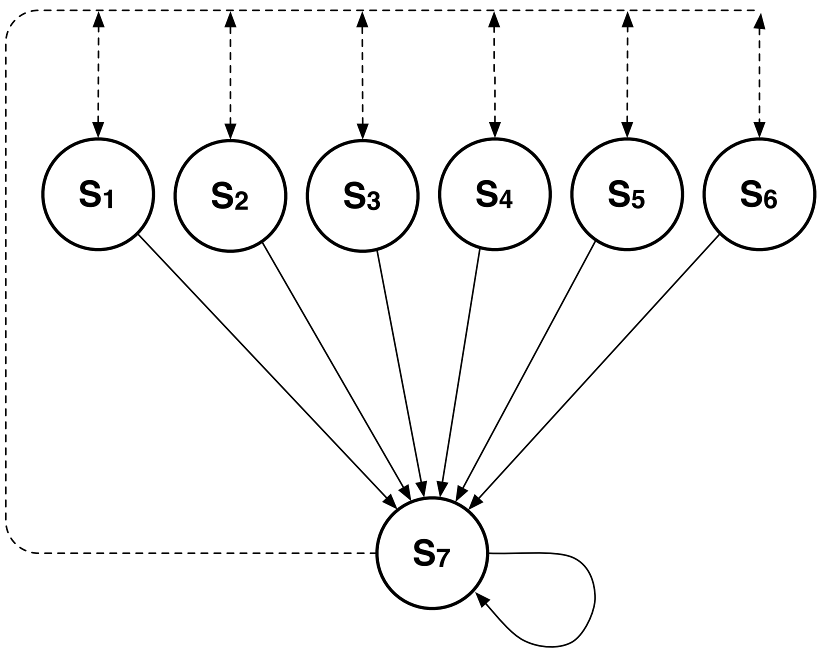

Figure 2.1: The environment and policies for Baird's Off-policy Counterexample. This counterexample shows that TD algorithms using function approximation are not guaranteed to converge on off-policy problems. In Baird's counter example, the target policy selects the solid line action with a probability of 1. The behaviour policy selects the solid line action with probability of 1/7. The behaviour policy selects the dotted line action with a probability of 6/7 (and the agent transitions to each of the six states indicated with probability of 1/6).

per-time-step complexity to $\mathcal{O}(kn)$, where k is a moderately chosen natural number, but still requires $\mathcal{O}\left(n^2\right)$ memory, which can be prohibitive in larger scale problems (Geramifard, Bowling, Sutton 2006).

Another attempt at solving off-policy problems with function approximation has been the residual gradient (RG) method (Baird, 1995). The residual gradient method is a stochastic gradient descent algorithm that minimizes the expected squared TD-error. Unfortunately, in order to sample the gradient on a given update, the RG method requires two independent samples of the next state. We refer to this requirement for two samples as *double sampling.* It is often difficult to obtain double samples unless the agent has a model of the environment. Baird suggests that one can ignore the need for double sampling and use only one sample for the next state, however, the solution found in this case has been shown to be different from the TD solution. Further more, when using function approximation, even if the agent is able to double sample, the solution found is still different from the TD solution.

In spite of all the previous work discussed above, there has long remained a need for TD methods which use function approximation that are convergent on off-policy problems while still maintaining a linear space and per-time-step complexity. In the next chapter, we will introduce the gradient-TD family of algorithms which address this long-standing need in the field.

Chapter 3

Gradient-TD Algorithms

In this chapter, we introduce the gradient-TD algorithms we will be looking at in this thesis. For our purposes, we use the name gradient-TD family to refer to all methods directly related to the relatively new GTD algorithm (Sutton, Szepesvári, & Maei 2009); accordingly, we are not referring to other gradient-descent based methods such as residual gradient methods or policy gradient methods. In particular, in this thesis we focus on three of the existing gradient-TD algorithms: GQ, GQ2, and GQ-NEU. For readers familiar with the gradient-TD literature, the names GQ-NEU and GQ2 will be unfamiliar however, the algorithms themselves shouldn't be: GQ-NEU and GQ2 are state-action versions of the GTD and GTD2 algorithms presented by Sutton, Szepesvári, and Maei, (2009) and Sutton, Maei, Precup, et al. (2009). GQ-NEU and GQ2 do not appear formally by name in any publications, though GQ-NEU is briefly described under the name GQE(0), as an extension to GTD by Sutton, Szepesvári, and Maei in their 2009 paper. GQ can be found originally in (Maei, Sutton 2010).

The family of gradient-TD algorithms is a set of stochastic gradient descent methods with convex objective functions that when minimized reach the TD solution for reinforcement learning problems. The gradient-TD family of algorithms is particularly interesting as all of its methods are guaranteed to converge on off-policy problems when using function approximation. Most excitingly, the gradient-TD algorithms only require linear space and computation-per-time-step time complexity, and avoid the need for double sampling, thus making them a simple light-weight solution methods for off-policy problems. As mentioned in chapter 2, remaining convergent on off-policy problems while using function approximation has long been an elusive goal in RL, thus gradient-TD's light-weight solution for off-policy problems is an exciting development.

Each gradient-TD method has three defining details which makes it unique. The first of these is the choice of objective function. The second characteristic is the particular derivation of the gradient from the objective function—while GQ2 and GQ have the same objective function, they behave differently in practice due to differences in how their gradients are derived. The third characteristic is how the gradient is approximated. Typically in stochastic gradient-descent methods the gradient can be sampled, however, all algorithms in the gradient-TD family discussed here have objective functions whose gradients' cannot be sampled in entirety without inducing bias. To avoid this, our gradient-TD methods use a trick where the gradient is divided into two pieces: one piece is sampled while the other piece is estimated by a learned approximation of that portion's expected value.

This work focuses on using linear function approximation, as such, the following derivations are written using linear function approximation and not for the more general case. Also, the following three algorithms are all state-action methods that make use of the following two notations: the state-action features $\phi(s_t, a_t)$ are abbreviated as ϕ_t and we use the notation $\bar{\phi}_t^\pi = \bar{\phi}^\pi(s_t) = \sum_a \pi(s_t, a)\phi(s_t, a)$ for the expected-next-action feature set under policy π (similarly, $\bar{\phi}_t^\mu$ is the expected-next-action feature set under policy μ).

We will present our gradient methods chronologically—we begin with GQ-NEU as its state-valued compliment, GTD, is the first gradient-TD method to have been developed. We then introduce GQ2 and GQ, again in the order of their original presentation in the literature. Before we can begin to look at the derivations of our gradient-TD methods, we begin with a mini-meta-discussion about the selection of objective functions.

3.1 An Argument for Bellman Error Based Objective Functions

In this section, we will briefly discuss several potential objective functions, and how they uniquely flavour learning, and discuss why we favour Bellman Error based objective functions.

Looking to the problem at hand, our algorithms are trying to solve the prediction problem of learning a value function to predict the expected return. Accordingly, a natural objective function might be the mean squared error (MSE)[1]:

[1]As this thesis exclusively uses algorithms that learn state-action values, we use the notation for the state-action value function Q^π, even when discussing concepts that are general to both state and state-action value functions

$$MSE(\theta) = \|\hat{Q}_\theta - Q^\pi\|_D^2 \tag{3.1}$$

Note, the norm $\|\cdot\|_D^2$ is the outer product of a vector weighted by the matrix D. In this case, D is a diagonal matrix whose diagonal values, $d_{s,a}$ correspond to the relative frequency that each state-action pair is visited under the behaviour policy.

As the MSE requires having the true value function, Q^π, to compare with, the MSE is not a feasible objective function for gradient methods. This poses a conundrum as the true value function is what we are trying to learn. Another similar potential objective function is the mean squared return error (MSRE). The MSRE uses a sample of the return in place of the true value function, however, this has the problem of having to wait for sampled returns to use the objective function for learning. It is important to be aware that both of these methods favour precision in the value function: when learning, they focus on making the value of individual state-action pairs as accurate as possible.

In contrast to MSE and MSRE are objective functions based on the Bellman Error, such as the Mean Squared Bellman Error (MSBE):

$$\text{MSBE}(\theta) = \|\hat{Q}_\theta - T_\pi^\gamma \hat{Q}_\theta\|_D^2$$

Here, T_π^γ is the Bellman operator, a matrix which projects the the value function one step forward.The MSBE has the benefit of being computable without knowledge of the true value function. Perhaps more subtle though, is the fact that the MSBE prioritizes self-consistency in the learned value function over precision in the value function. While the solution of the MSE, MSRE, and MSBE are ultimately all the same when we are not using function approximation, this difference in priorities—favouring a self-consistent value function versus a precise value function—impacts how the value function evolves as it is being learned. In RL, it is generally acknowledged that a self-consistent value function is more effective in practice for the full RL problem with control. As such, the gradient-TD methods we present here all use Bellman error based objective functions. We will return to this thought again in chapter 4 when discussing our performance measure for experiments.

3.2 Deriving GQ-NEU

GQ-NEU uses the norm of the expected update (NEU) as it's objective function

$$NEU(\theta) = \mathbb{E}\left[\delta\phi\right]^\top \mathbb{E}\left[\delta\phi\right] \tag{3.2}$$

Here δ is the TD-error using linear function approximation, state-action features, and the expected next action notation introduced in chapter 2: $\delta_t = r_{t+1} + \gamma\theta_t^\top \bar{\phi}_{t+1}^\pi - \theta_t^\top \phi_t$. Note this is the TD-error as used by Expected Sarsa as described in chapter 2. The NEU is exactly what it claims to be: it is the 2-norm of the expected TD-update. As an objective function, the NEU has many desirable properties: it is quadratic and unimodal and it achieves its minimum at the TD-solution. Also, the TD-error is a sample based expression of the Bellman error, meaning the NEU is a Bellman error based objective function.

The gradient of the NEU is:

$$\begin{aligned}
\nabla_\theta NEU(\theta) &= 2(\nabla_\theta \mathbb{E}\left[\delta\phi\right])\, \mathbb{E}\left[\delta\phi\right] \\
&= 2\,\mathbb{E}\left[\phi(\nabla_\theta\delta)^\top\right]^\top \mathbb{E}\left[\delta\phi\right] \\
&= -2\,\mathbb{E}\left[\phi_t(\phi_t - \gamma\bar{\phi}_{t+1}^\pi)^\top\right]^\top \mathbb{E}\left[\delta_t\phi_t\right]
\end{aligned} \tag{3.3}$$

In stochastic gradient-descent approaches the gradient is sampled and θ is updated, moving in the direction opposite to the gradient:

$$\theta_{t+1} = \theta_t - \alpha\nabla_\theta NEU(\theta) \tag{3.4}$$

Unfortunately, the gradient of the NEU is a product of two related expected values. We cannot use samples for both expected values without biasing the gradient. To avoid this bias, one of the two expected terms can be sampled while the other can be approximated with a learned estimate. At this point, there are two options: estimate the first expectation $\mathbb{E}\left[\phi_t(\phi_t - \gamma\bar{\phi}_{t+1}^\pi)^\top\right]^\top$ or the second expectation $\mathbb{E}\left[\delta_t\phi_t\right]$. Estimating the first expectation would involve learning a matrix of size n^2 (where n is the number of features in ϕ). To avoid the larger space complexity required to store an estimate of the first term, GQ-NEU samples the first expectations. GQ-

NEU learns a vector of weights w such that $w = \mathbb{E}[\delta\phi]$ when converged to estimate the remaining portion of the gradient. A simple supervised learning update is used to learn w. GQ-NEU uses its approximation of w in conjunction with samples to approximate the gradient. The gradient is then used in a standard stochastic gradient descent update for learning θ. The updates for θ and w for GQ-NEU are as follows in equations 3.5 and 3.6:

$$\theta_{t+1} = \theta_t + \alpha_t(\phi_t - \gamma\bar{\phi}^{\pi}_{t+1})\phi_t^{\top} w_t \tag{3.5}$$

and

$$w_{t+1} = w_t + \beta_t(\delta_t\phi_t - w_t) \tag{3.6}$$

Note that the constant factor of 2 from the gradient can be dropped. α_t and β_t are standard positive valued step-size parameters.

3.3 The Projected Bellman Error and the Derivation of GQ2 and GQ

GQ2 and GQ are state-action value counterparts to GTD2 and TDC, which were both developed to find a faster gradient-TD algorithm than GTD (Sutton, Maei, Precup, et al. 2009). We present GQ2 and GQ together as they share one crucial defining element—-their objective function. Both algorithms use the mean-square projected Bellman error (MSPBE) as their objective function.

3.3.1 The Projected Bellman Error

There is one potential problem with the mean-square Bellman error which we did not mention above. Ideally, when the value function is exactly learnable, an algorithm minimizing the mean-square bellman error reaches the fixed point where $\hat{Q} = T^{\gamma}_{\pi}\hat{Q}$. Unfortunately, when we are using function approximation, $T^{\gamma}_{\pi}\hat{Q}$ may not be representable by our function approximation method. As such, it may be impossible to reach the fixed point of the MSBE. The mean-square projected Belman error (MSPBE) is a response to this limitation of the MSBE. The MSPBE introduces a projection operator , Π, which will take any value function q and project it onto the nearest value function that is representable by our function approximator:

$$\Pi q = \hat{Q}_\theta \text{ where } \theta = \underset{\theta}{\operatorname{argmin}} \|\hat{Q}_\theta - q\|_D^2$$

With this projection operator, we can define a Projected Bellman Equation:

$$\hat{Q}_\theta = \Pi T \hat{Q}_\theta$$

From here, we can naturally define the mean-square projected Bellman error as follows:

$$MSPBE(\theta) = \|\hat{Q}_\theta - \Pi T \hat{Q}_\theta\|_D^2 \tag{3.7}$$

The advantage of the MSPBE is that methods using function approximation can always reach the point where the MSPBE is 0. This is not necessarily true of the MSBE when using function approximation. While there is no strong general claim that can be made about the comparative quality of the solution obtained by minimizing the MSPBE versus minimizing the MSBE, the guarantee that the minimum of the MSPBE is attainable is a desirable quality in an objective function.

3.3.2 Deriving GQ and GQ2

Before we derive GQ and GQ2 using the MSPBE, first let us establish some relevant and necessary equalities, and then rewrite the MSPBE in terms that will be easier to work with. First, when using a linear value function— $\hat{Q}_\theta = \Phi\theta$, where Φ is the matrix whose rows are the feature vectors for each state or state-action pair—the projection operator Π is linear and independent of θ and expressed as:

$$\Pi = \Phi(\Phi^\top D \Phi)^{-1} \Phi^\top D \tag{3.8}$$

Second, the following expectations can be expressed in vector-matrix quantities as shown:

$$\mathbb{E}\left[\phi\phi^\top\right] = \sum_{s,a} d_{s,a} \phi_{s,a} \phi_{s,a}^\top = \Phi^\top D \Phi,$$

$$\begin{aligned}\mathbb{E}\left[\delta, \phi\right] &= \sum_{s,a}\left(R_{s,a} + \gamma \sum_{s',a'} P^{a}_{s,s'}\pi(a' \mid s')\hat{Q}_\theta(s'a, a) - \hat{Q}_\theta(s,a)\right) \\ &= \Phi^\top D(T\hat{Q}_\theta - \hat{Q}_\theta),\end{aligned}$$

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

Given these relationships, we can now rewrite the MSPBE to be in terms of expectations:

$$\begin{aligned}MSPBE(\theta) &= \|\hat{Q}_\theta - \Pi T\hat{Q}_\theta\|^2_D \\ &= \|\Pi(\hat{Q}_\theta - T\hat{Q}_\theta)\|^2_D \\ &= (\Pi(\hat{Q}_\theta - T\hat{Q}_\theta))^\top D(\Pi(\hat{Q}_\theta - T\hat{Q}_\theta)) \\ &= (\hat{Q}_\theta - T\hat{Q}_\theta)^\top \Pi^\top D\Pi(\hat{Q}_\theta - T\hat{Q}_\theta) \\ &= (\hat{Q}_\theta - T\hat{Q}_\theta)^\top D^\top \Phi(\Phi^\top D\Phi)^{-1}\Phi^\top D(\hat{Q}_\theta - T\hat{Q}_\theta) \\ &= (\Phi^\top D(T\hat{Q}_\theta - \hat{Q}_\theta))^\top(\Phi^\top D\Phi)^{-1}\Phi^\top D(T\hat{Q}_\theta - \hat{Q}_\theta) \\ &= \mathbb{E}\left[\delta\phi\right]^\top \mathbb{E}\left[\phi\phi^\top\right]^{-1}\mathbb{E}\left[\delta\phi\right] \qquad (3.9)\end{aligned}$$

Note here that the MSPBE only differs from the NEU (equation 3.2) by the inverse of the expected feature-covariance matrix. From here, we can derive the gradient of the MSPBE:

$$-\frac{1}{2}\nabla MSPBE(\theta) = \mathbb{E}\left[(\phi_t - \gamma\bar{\phi}^\pi_{t+1})\phi_t^\top\right]\mathbb{E}\left[\phi_t\phi_t^\top\right]^{-1}\mathbb{E}\left[\delta_t\phi_t\right] \qquad (3.10)$$

It is at this point where GQ2 and GQ diverge. GQ2 uses the gradient as written in equation 3.10, where GQ rephrases the gradient in a slightly different form :

$$
\begin{aligned}
-\frac{1}{2}\nabla MSPBE(\theta) &= \mathbb{E}\left[(\phi_t - \gamma\bar{\phi}^{\pi}_{t+1})\phi_t^{\top}\right]\mathbb{E}\left[\phi_t\phi_t^{\top}\right]^{-1}\mathbb{E}\left[\delta_t\phi_t\right] \\
&= \left(\mathbb{E}\left[\phi_t\phi_t^{\top}\right] - \gamma\,\mathbb{E}\left[\bar{\phi}^{\pi}_{t+1}\phi_t^{\top}\right]\right)\mathbb{E}\left[\phi_t\phi_t^{\top}\right]^{-1}\mathbb{E}\left[\delta_t\phi_t\right] \\
&= \mathbb{E}\left[\delta_t\phi_t\right] - \gamma\,\mathbb{E}\left[\bar{\phi}^{\pi}_{t+1}\phi_t^{\top}\right]\mathbb{E}\left[\phi_t\phi_t^{\top}\right]^{-1}\mathbb{E}\left[\delta_t\phi_t\right] \qquad (3.11)
\end{aligned}
$$

Both of these two expressions of the gradient suffer from the same problem that we also saw with the gradient of the NEU: they are made up of the product of several related expectations, and we cannot use samples to approximate all three expectations without introducing bias to our gradient. To avoid this, both GQ2 and GQ use the same trick as GQ-NEU and learn a set of linear weights, w to approximate a portion of the gradient:

$$
w = \mathbb{E}\left[\phi_t\phi_t^{\top}\right]^{-1}\mathbb{E}\left[\delta_t\phi_t\right] \qquad (3.12)
$$

The choice of which expectations to approximate and which to sample is again made to avoid having to store a matrix of size n^2, where n is the number of features. Note that w is in the form of a least squares estimate, and so we can learn w using a standard least squares update:

$$
w_{t+1} = w_t + \beta_t(\delta_t - \phi_t^{\top}w_t)\phi_t. \qquad (3.13)
$$

Using w and the two forms of the MSPBE gradient, we can define our updates for GQ2 and GQ as follows:

$$
\theta^{GQ2}_{t+1} = \theta^{GQ2}_{t} + \alpha_t(\phi_t - \gamma\bar{\phi}^{\pi}_{t+1})(\phi_t^{\top}w_t), \qquad (3.14)
$$

and

$$
\theta^{GQ}_{t+1} = \theta^{GQ}_{t} + \alpha_t\delta_t\phi_t - \alpha_t\gamma\bar{\phi}^{\pi}_{t}(\phi_t^{\top}w_t). \qquad (3.15)
$$

The GQ update has one unique property particularly worth noting at this point: the update has the form of an Expected Sarsa update with an added “correction” term that adjusts the update to follow the gradient of the MSPBE objective function.

As such, when $w = 0$, GQ does a typical standard TD style update. Also worth mentioning is the fact that when θ has converged for both GQ2 and GQ, w will converge to zero.

Chapter 4

Assessment on On-Policy Prediction Problems

With this chapter, we begin the experiments central to this thesis. The experiments in this thesis are structured as a series, which incrementally modify the problem and algorithms. In this chapter, we introduce the modified Random Walk with actions environment which we will be using throughout this thesis. We then begin with an on-policy experiment to determine whether it is more effective to use Sarsa or Expected Sarsa as the representative conventional-TD algorithm for our experiments. Finding Expected Sarsa to be the faster and more robust algorithm of the two on this problem, we perform a second experiment comparing Expected Sarsa and our gradient-TD algorithms, GQ, GQ2 and GQ-NEU. This experiment gives the reader a simple, uncomplicated look at the relative convergence speeds of our gradient-TD methods, and affirms existing results which show that gradient-TD algorithms may be slower than conventional-TD methods (Maie 2011). With this established, we begin our search for faster gradient-TD algorithms by introducing new algorithms, and discuss the convergence speed and parameter sensitivity of our current gradient-TD methods.

4.1 Random Walk with Actions Problem

For clarity, allow us to begin by making the distinction between our usage of the *environment* and the *problem* in our discussion of the experiments in this thesis. The environment refers to the markov decision process (MDP) that provides the set of available states and actions, and the transition functions necessary to make up the world dynamics. For the purposes of this thesis, a problem is more generally thought of as the question one wants to ask; it includes the environment, but also entails all the detailed settings of environment variables that can greatly change the

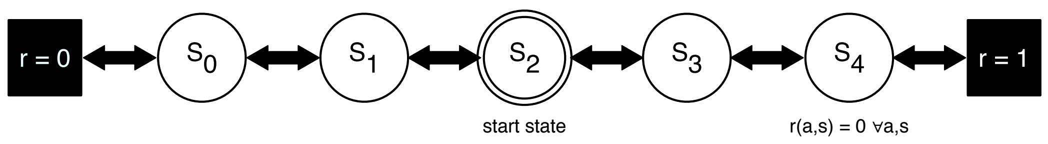

Figure 4.1: The 5 state markov chain environment

task at hand. In this thesis all the experiments use the same environment but each experiment features different problems to investigate our algorithms' ability to learn in varied situations.

The environment we will be using is a variation of the random walk environment (Sutton, Maei, Precup, et al. 2009; Maei 2011) which is modified to give the agent the ability to select actions. For the sake of brevity, we will also call this environment the random walk environment, due to it's relation to the original random walk environment. To begin with the environment, a visual depiction of the random walk environment can be seen in figure 4.1. There are 5 states, with 2 absorption states, which form a chain. The environment is episodic, where an episode ends whenever an agent enters one of the two edge absorption states. Every non-absorption state provides the agent with two possible actions: move left or move right. Movement is deterministic so that choosing to move left always results in the agent moving to the next state to its left, and accordingly, choosing the move right action always moves the agent to the right. As such, all the randomness in the experiment comes from an agent's behaviour policy. The reward signal for the random walk environment is 0 in all states except for the right absorption state, where the agent receives a reward of 1 before termination of the episode.

The general form of the problems being tasked to our algorithms in this chapter is to learn a linear state-action value function for the random-walk environment when on-policy, using a fixed equally random behaviour/target policy. We look at three different state-action feature sets, giving us three unique on-policy learning problems for this experiment. The three feature sets, *tabular* features, *inverted* features, and *dependent* features (shown in table 4.1) have been selected for their varying degrees of difficulty.

The tabular feature set are the least complex of the three feature sets: each state-action pair's feature vector has only one non-zero element and no two feature vectors share a non-zero element. As a result, there is no generalizing or aliasing between state-action pairs. The tabular features are equivalent to the table-look up case and require no approximation—the exact value of each state is learned.

state	tabular	inverted	dependent
s_0, a_0	1 0 0 0 0 0 0 0 0 0	0 0.5 0.5 0.5 0.5 0 0 0 0 0	1 0 0 0 0 0
s_1, a_0	0 1 0 0 0 0 0 0 0 0	0.5 0 0.5 0.5 0.5 0 0 0 0 0	$\frac{1}{\sqrt{2}}$ $\frac{1}{\sqrt{2}}$ 0 0 0 0
s_2, a_0	0 0 1 0 0 0 0 0 0 0	0.5 0.5 0 0.5 0.5 0 0 0 0 0	$\frac{1}{\sqrt{3}}$ $\frac{1}{\sqrt{3}}$ $\frac{1}{\sqrt{3}}$ 0 0 0
s_3, a_0	0 0 0 1 0 0 0 0 0 0	0.5 0.5 0.5 0 0.5 0 0 0 0 0	0 $\frac{1}{\sqrt{2}}$ $\frac{1}{\sqrt{2}}$ 0 0 0
s_4, a_0	0 0 0 0 1 0 0 0 0 0	0.5 0.5 0.5 0.5 0 0 0 0 0 0	0 0 1 0 0 0
s_0, a_1	0 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0.5 0.5 0.5 0.5	0 0 0 1 0 0
s_1, a_1	0 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0.5 0 0.5 0.5 0.5	0 0 0 $\frac{1}{\sqrt{2}}$ $\frac{1}{\sqrt{2}}$ 0
s_2, a_1	0 0 0 0 0 0 0 1 0 0	0 0 0 0 0 0.5 0.5 0 0.5 0.5	0 0 0 $\frac{1}{\sqrt{3}}$ $\frac{1}{\sqrt{3}}$ $\frac{1}{\sqrt{3}}$
s_3, a_1	0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 0.5 0.5 0.5 0 0.5	0 0 0 0 $\frac{1}{\sqrt{2}}$ $\frac{1}{\sqrt{2}}$
s_4, a_1	0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0.5 0.5 0.5 0.5 0	0 0 0 0 0 1

Table 4.1: This table shows the three types of features used in the random walk problem. Note that the feature vector for s_i, a_0 and s_i, a_1 are related: the feature vector can be thought of having two halves corresponding to the two actions, and the first half of s_i, a_0 is equal to the second half of s_i, a_1

The inverted feature set is thusly named because it is the result of taking the tabular feature vector for a state-action pair, inverting it, so that all 0's are turned to 1's and all 1's changed to 0's, and normalizing the vector. The inverted feature set still forms an independent set—one can still exactly solve the value function with these features—but there are a large number of shared features between states, making it significantly harder to learn the true value function using the inverted features than when using the tabular features.

Lastly, the dependent feature set was constructed as a feature set that no longer forms an independent set. As a result, an agent may not be able to learn the true state-action value function. Note that the dependent feature vectors all have a norm of 1, just as the tabular and inverted features. Ensuring the norm is 1 means there is no question of magnitude effects causing differences in performance when using these three feature sets.

4.2 Experimental design

In this experiment, we apply our gradient-TD and conventional-TD methods to versions of the random walk with actions problem with various feature representations. To fully specify our experiment, there are several algorithmic and environment variables which must be set. The algorithms have step-size parameters—our conventional-TD methods, Sarsa and Expected Sarsa, have α, and our gradient-TD algorithms have both α and β. The performance of an algorithm can be quite sensitive to the value of these step-size parameters and so it is important to try each algorithm with a wide range of parameter values.

For α, preliminary experimentation showed that it is important to try both small and large values up to 2; larger values always performed worse or diverged. The specific set of α values used for all algorithms can be found in table 4.2.

The selection of β is largely dependent on the value of α an algorithm is using—a value of β may be large when used with a small α, but may be much too small when used with a large α. The relationship between α and β is extremely important. Accordingly, one would want to try a different set of β parameters for $\alpha = 0.001$ than for $\alpha = 2.0$. To avoid having multiple tables of β values used, and to simplify the reporting of our results, we introduce a variable η that we use to compute β: $\beta = \alpha * \eta$. The practice of using the parameter η to relate α to β was also done in (Sutton, Maei, Precup, et al. 2009). The values of η used for this experiment were $\{0.1, 0.25, 0.5, 0.75, 1, 2, 4\}$. An experiment was run using the gradient-TD algorithms for every combination of α and η. As a result, every α value was evaluated using a set of proportionally smaller and larger values of β. A summary table of all the α and η values can be found in table 4.2.

α	0.001	0.003	0.005	0.008	0.01	0.025	0.05	0.1	0.15	0.2
	0.25	0.5	0.9	1.0	1.2	1.4	1.6	1.8	2.0	
η	0.1	0.25	0.5	0.75	1	2	4			

Table 4.2: A table of agent parameters used where the gradient-td second learning rate β is a scalar multiple of α where ($\beta = \alpha * \eta$). Each unique α and η pairing were tested.

One final algorithmic variable affecting the performance of an algorithm is the initial values of its learned components—θ for all our algorithms and w for our gradient-TD algorithms. In a control experiment, where the agents actions can depend on the current estimate of the value function, the initial setting of θ can have a large impact on the performance of an algorithm. As all the experiments in this thesis use fixed policies, the initial value of θ is less of a concern than in control problems. In this experiment, we initialize θ to be a vector where all the weights are 0.5. Given the limited work done with the gradient-TD algorithms to date, there exists little information about how the initial values of w affect the algorithms and how other parameters affect the initialization of w. We chose to initialize w to be a vector of zeros.

Every instance of our algorithms with unique parameters was run for 30 runs of 1000 episodes each using each feature vector on each of the three random walk problems. At the beginning of every run, the agent's weights were reset to their initialization values—and so each run is a fresh experiment of 1000 episodes. At the beginning of every episode, the agent was placed in the middle state, (s_2) and ran until the agent reached one of the two terminal states on either end of the environment. To measure the performance of our algorithms, we recorded the root mean square projected bellman error (abbreviated as RMSPBE) of the weight vector θ at the end of every episode. Recall that the mean square projected bellman error is the objective measure minimized by GQ2 and GQ. A definition and explanation of the root mean squared projected bellman error can be found in chapter 3.

Traditionally, much work has been done using the mean squared error ($\mathrm{MSE}(\theta) = \sum_{s\in S} D(s) \sum_{a\in Actions} \pi(a \mid s)\|Q^{\pi}(s,a) - \hat{Q}_t(s,a)\|^2$) or the mean squared Bellman error ($\mathrm{MSBE}(\theta) = \|\hat{Q}_\theta - T_\pi^\gamma \hat{Q}_\theta\|_D^2$) as performance measures. Before we discuss why we chose to use the RMSPBE, let us first address the differences between the MSBE/RMSPBE and the MSE. Ultimately, the Bellman based errors and the MSE have the same solution if the value function can be represented exactly. Unfortunately, this is not always the case when using function approximation and so it is important to be mindful of the differences between these measures. The MSBE and the RMSPBE encourages a more self-consistent value function, penalizing discrepancies between related values. Alternately, the MSE encourages a more precise value function, focusing on making each individual state-action value accurate, without concern for the relationship between various state-action pairs. It is widely recognized that for the RL problem, it is often more useful to make the value function transitions more consistent than to make the value function more precise. It is the general view that the Bellman based errors are more appropriate error measure of the two for RL problems (Sutton 1984).

From there, let us look at the difference between the MSBE, and the RMSPBE. Both measures, being based on the Bellman equation, look at the difference between the value function and the value function for the next step—the difference between $\hat{Q}$ and $T_\pi^\gamma \hat{Q}$. What separates the Bellman error from the projected Bellman error, is that the projected bellman error insists that the next step value function be representable by our function approximator. When the value function is transformed by the Bellman operator T_π^γ, the resulting next step value function may not be representable by our function approximation method of choice. The projected Bellman error tries to "correct" for this by using the projection operator Π to project the next step value function back onto the closest representable value function. This is an appealing property as it means that when an algorithm has converged, it should have an RMSPBE of 0. This makes interpreting the performance of these algorithms easy—there is no question about what the converged value of a given function approximation method should be on a given problem. This is why we ultimately chose to use RMSPBE as our performance measure.

4.3 Conventional-TD Methods

One of the claims that we seek to investigate in this thesis is whether our gradient-TD methods are slower than conventional-TD methods. For the purpose of this work, we do not attempt an exhaustive comparison to all conventional-TD methods. Rather, we select a representative of conventional-TD methods which perform well. Our two selected candidates for a representative conventional-TD method are Sarsa and Expected Sarsa. A brief introduction to both these algorithms can be found in chapter 2. We chose Sarsa as it is a foundational state-action value TD method. We chose Expected Sarsa for two reasons. Firstly, Expected Sarsa has been shown to reduce variance in updates, allowing for the use of larger step-size (α) values in practice. This in turn can speed up learning (van Seijen, et al. 2009). Secondly, Expected Sarsa has a close relationship with GQ. When GQ is using a second step-size (β) of zero, and has initialized its second weight vector w to zero, the algorithm has the same update as Expected Sarsa.

We ran Sarsa and Expected Sarsa on the experiment described above. In figure 4.2, it can be seen that when using the best parameter setting (where the best parameters have the lowest RMSPBE averaged over all 30 runs of 1000 episodes), Expected Sarsa is significantly faster than Sarsa. In figure 4.3, we see that Expected Sarsa outperforms Sarsa regardless of the parameter settings and is more robust to parameter settings than Sarsa is on this problem. As Expected Sarsa appears to have the faster and more stable performance on this problem, we will use Expected Sarsa as our conventional-TD representative method for the remainder of this thesis.

4.4 Learning Rate of Conventional- and Gradient-TD Methods

Having settled upon Expected Sarsa as our conventional-TD method for our experiments, we then ran GQ, GQ2, and GQ-NEU through our random walk experiment. Figure 4.4, shows the trajectory of the RMSPBE error for the first 200 episodes for

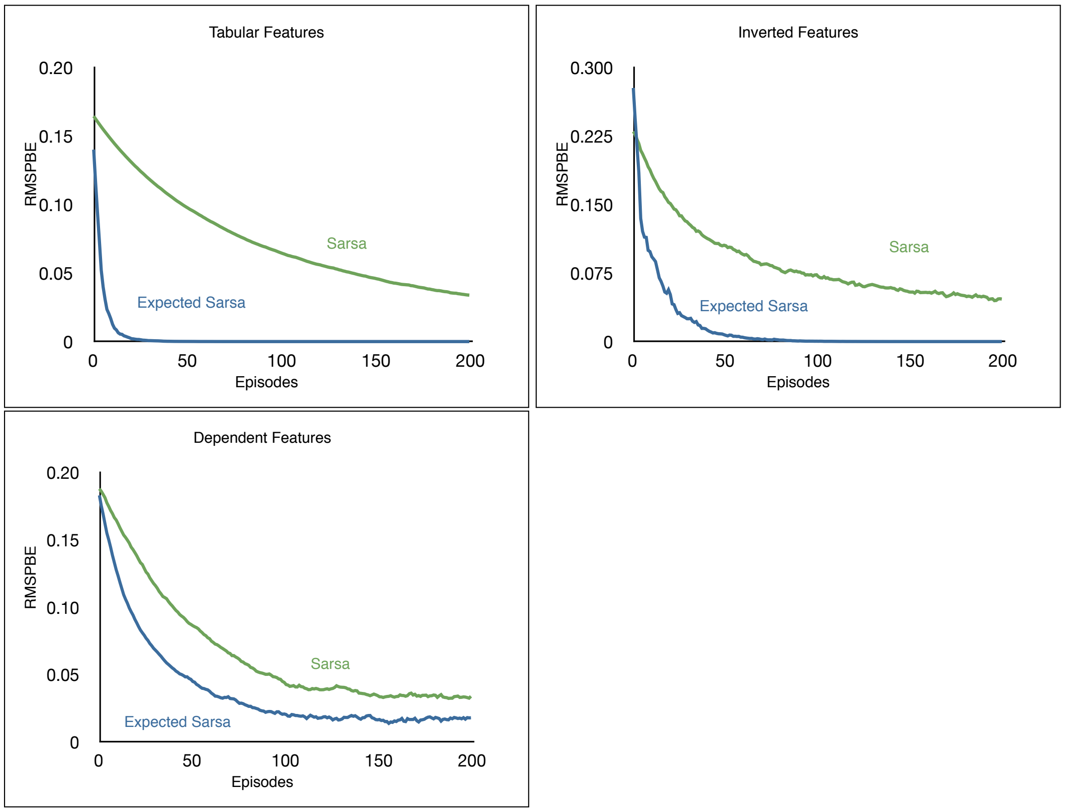

Figure 4.2: Graphing the RMSPBE of Sarsa and Expected Sarsa for the first 200 episodes. Both algorithms use their best α parameter for each of the three feature sets.

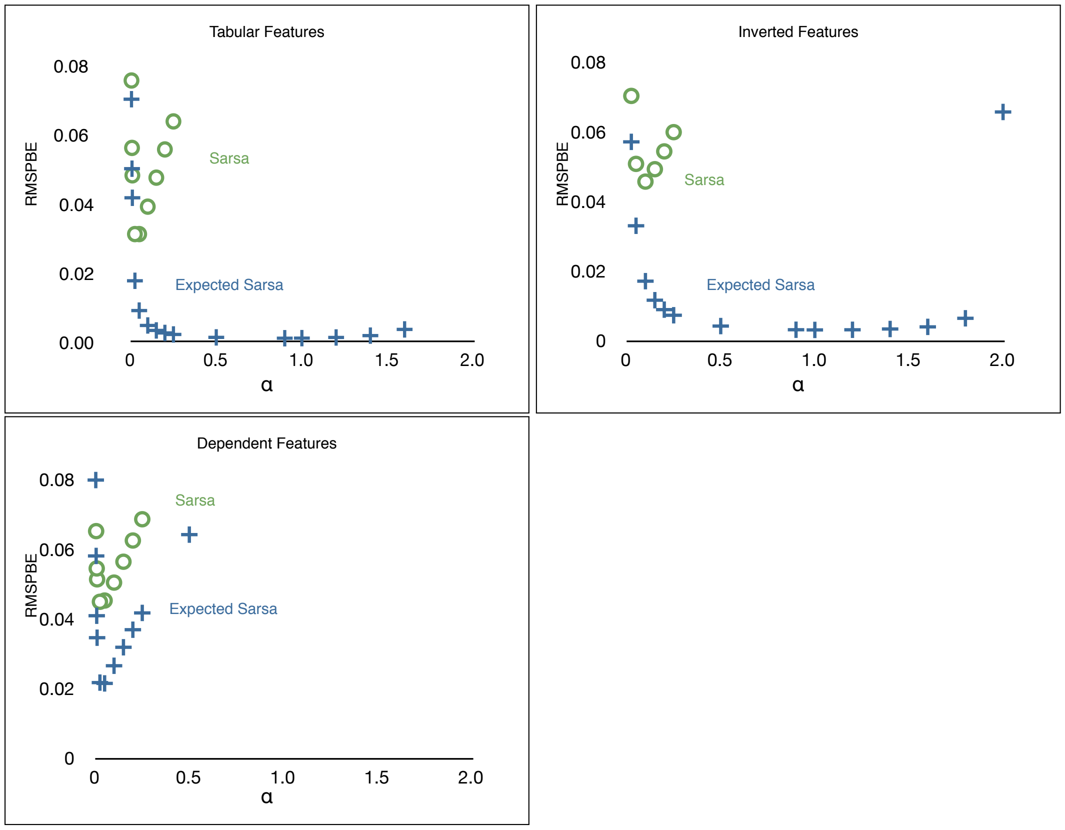

Figure 4.3: Graphing the RMSPBE of Sarsa and Expected Sarsa averaged over 1000 episodes when varying the setting of the step-size parameter α.

the gradient-TD algorithms and Expected Sarsa when using their best parameters for each of the three feature representations. These graphs show a clear ordering of the basic algorithms from slowest to fastest: GQ-NEU, GQ2, GQ, and Expected Sarsa. This result unsurprisingly mirrors results which were first presented by Sutton, Maei, Precup, et al. (2009), and again appear in Maei's thesis (Maei 2011). These earlier results showed an ordering of strict improvement when comparing GTD, GTD2, TDC and TD at learning state value functions on the no-actions version of random walk problem used in this experiment. The reader may recall that GTD, GTD2, TDC and TD are state-value function relatives of GQ-NEU, GQ2, GQ, and Expected Sarsa respectively and so it is unsurprising that our results show the same trends as the earlier work using the state-value based algorithms. While this experiment is hardly conclusive, it does add to previous evidence that suggests GQ is the fastest of our gradient-TD algorithms, and may be our best starting point for developing faster gradient-TD algorithms. This data also lends support to the idea that gradient-TD methods are currently slower than conventional-TD methods on on-policy problems. That our gradient-TD methods are indeed slower than conventional-TD methods further motivates our goal of finding faster gradient-TD algorithms.

4.5 Recomputed TD-error

Having looked at our three base gradient-TD algorithms, we now begin the quest for faster gradient-TD methods. Our first step towards this goal is a variation that can be applied to all our gradient-TD methods called the recomputed TD-error trick. The TD-error is an integral part of most TD algorithms, as discussed in chapter 2. The recomputed TD-error trick exploits the fact that all our gradient-TD methods use the TD-error (notated as δ in equations) in the updates for both of their primary weight vectors θ and w. The typical update for our gradient-TD methods involves computing δ and then update both θ and w. Note however, that δ depends on θ, and thus it may be advantageous to recompute δ after updating θ but before updating w, to allow the algorithm to take full advantage of the most up-to-date value estimates. It is this minor tweak which is what we will refer to as the recomputed TD-error trick. To test how useful this seemingly small change is, we ran the previous experiment again, using recomputed TD-error variants of our three gradient-TD algorithms (we add a plus sign to the end of the names of our algorithms when using the recomputed TD-error trick, thus we re-ran our experiments using the algorithms GQ+, GQ2+, and GQ-NEU+).

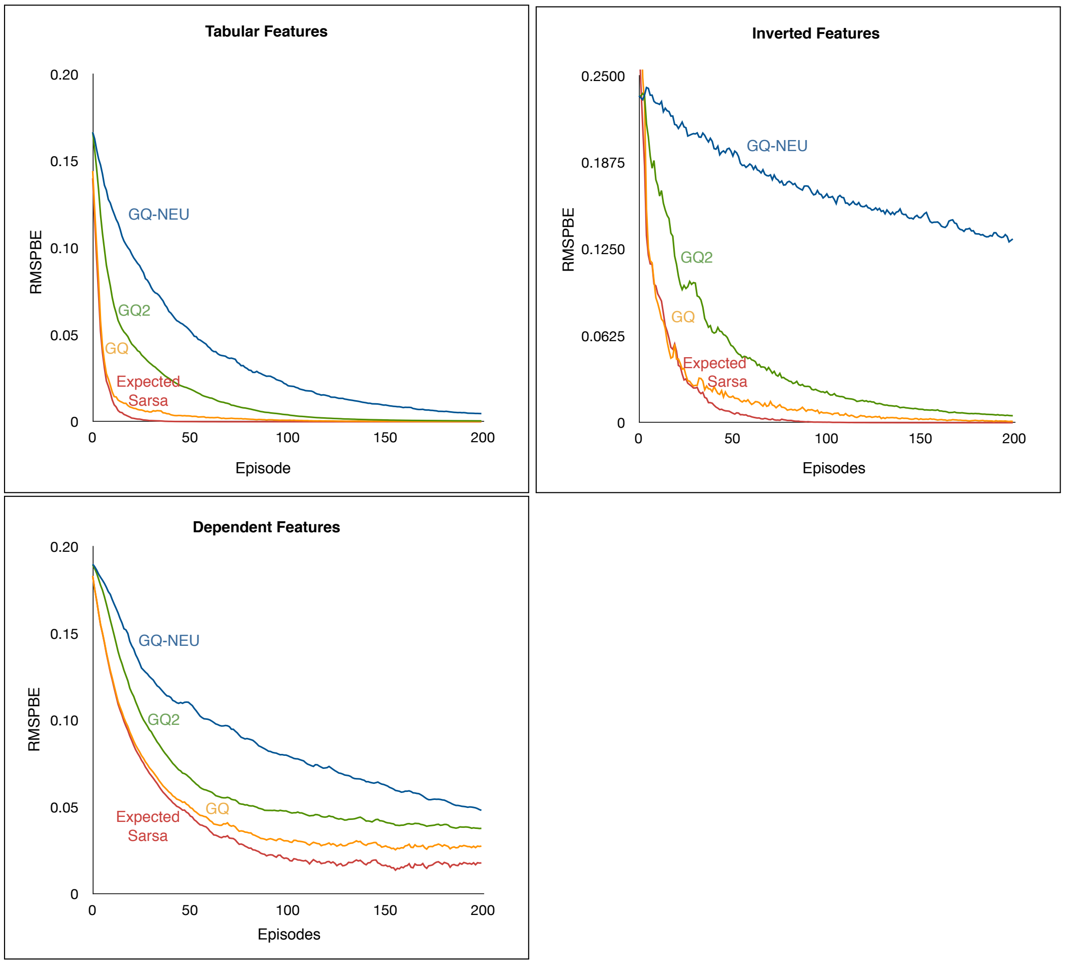

Figure 4.4: Graphing the average RMSPBE of the gradient-TD algorithms and Expected Sarsa for the first 200 episodes

In figure 4.5 we again show the error trajectory for the first 200 episodes for the gradient-TD algorithms and Expected Sarsa when using their best parameters, this time including our new gradient-TD variations that use the recomputed TD-error trick. Perhaps the most important thing to note here is that in the case of GQ2 and GQ there were no negative consequences to recomputing the TD-error in this experiment. In fact, GQ+ shows such a benefit as to achieve a performance level which is not statistically significantly different from that of Expected Sarsa on the problems using tabular and inverted features. It is only in the case of the dependent features that we see no improvement from GQ+. While recomputing the TD-error is a very simple modification of our original gradient-TD methods, it provide a non-trivial speed up on this simple problem. These results are of course very limited—it is not clear that this effect will continue to be significant on larger more difficult problems, or off-policy problems—but it is promising to see noticeable improvement from such a small change to the algorithms.

There is one major thing which must not be overlooked when comparing our gradient-TD methods with Expected Sarsa: our gradient-TD methods have the disadvantage of having two step-size parameters which must be tuned (α and η), over Expected Sarsa's one (α). In figure 4.6, we can see how GQ+'s performance is affected by the selection of its parameters α and η(note that we omit showing the same graphs for GQ as the results are very similar). What is particularly concerning in this figure is how erratic the relationship between α and η can be: while there are values of α for which the value of η is inconsequential, often the best performing α values are very sensitive to η. A slight change to η may result in a drastic change in performance. The unfortunate conclusion that we can draw from this is that our gradient-TD methods may require great care when setting the second learning weight η in order to ensure good performance.

One other thing to note about these results is that GQ+ consistently performs best when η is small in this experiment. This result is unsurprising if one recalls that when β is 0, because our weight vector w is initialized to 0, GQ and GQ+ are the same as Expected Sarsa. Thus it makes sense that our performance is closest to Expected Sarsa, which we have already shown to be faster than our gradient-TD methods, when η is small. For GQ, the second weight vector w contributes a "correction term" to our updates of θ, and so in an off-policy problem where a conventional-TD method would be divergent, having too small of an η value can have disastrous results: if η is small, and w is not learned quickly, an agent may become numerically unstable. The natural concern from this then is that the optimal value of η may be extremely different for on-policy and off-policy problems. This sensitivity

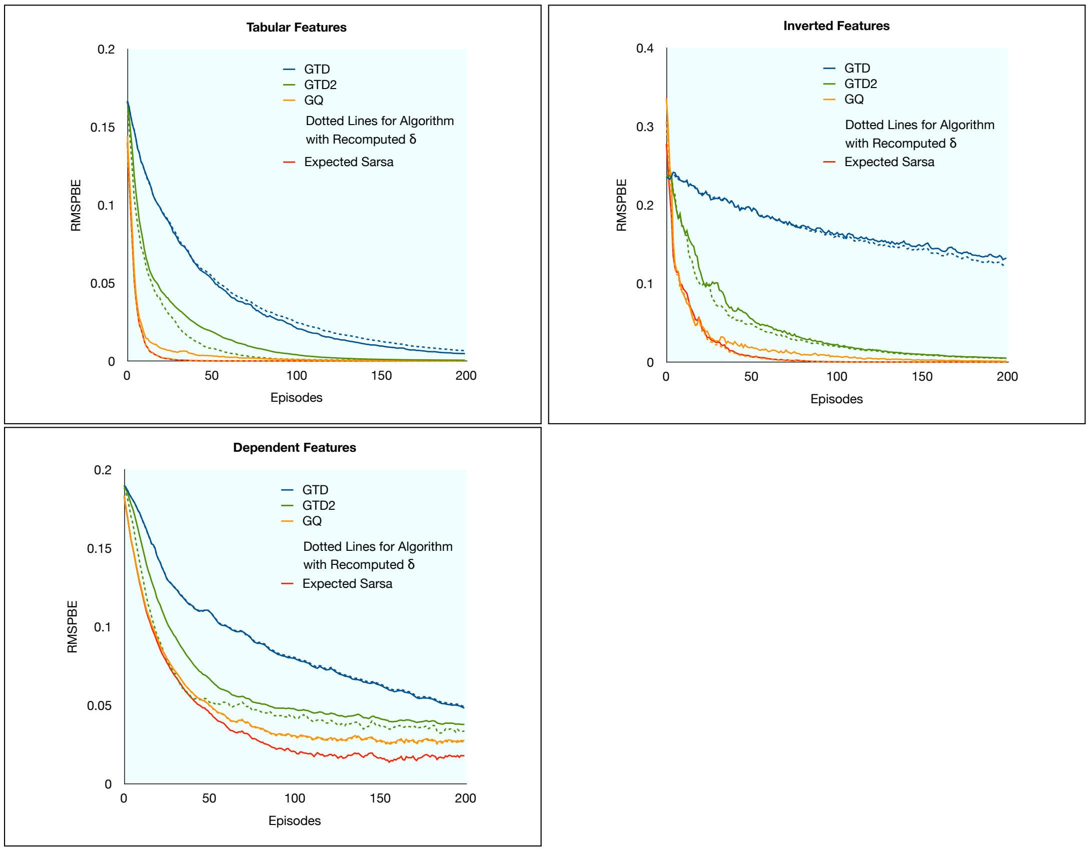

Figure 4.5: Graphing the average RMSPBE of the gradient-TD algorithms with their recomputed TD-error variants and Expected Sarsa for the first 200 episodes

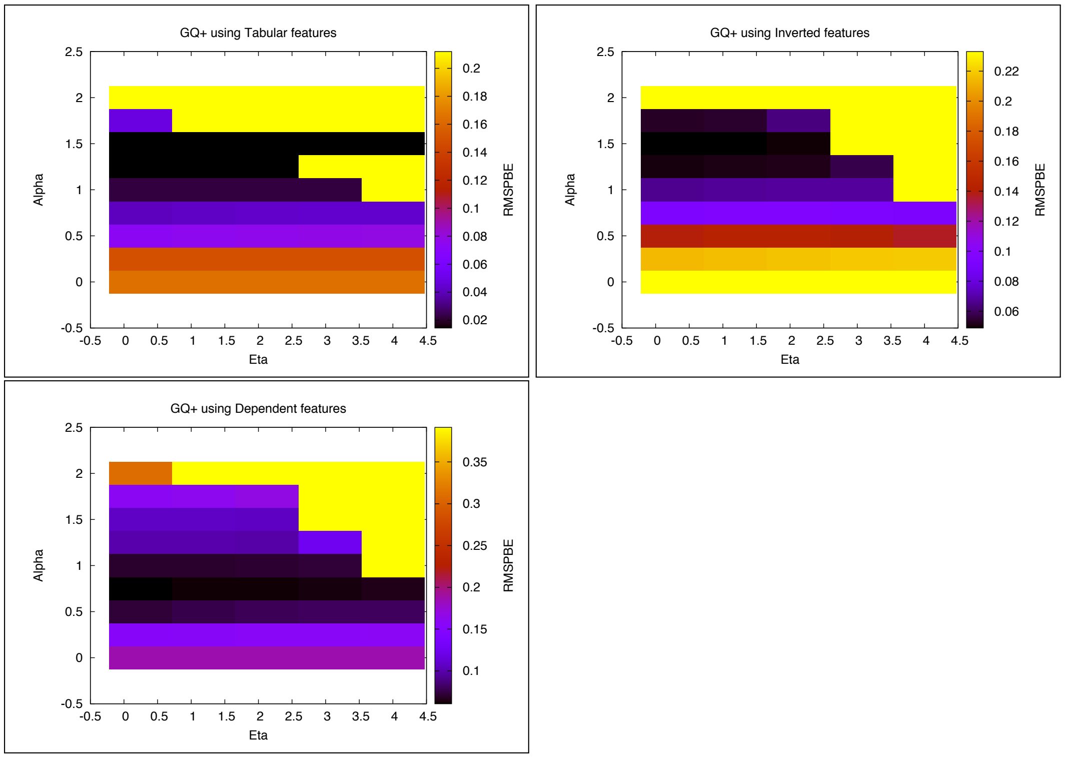

Figure 4.6: Graphing the effect changing α and η have on the averaged RMSPBE of GQ+. For some of the high values of α, where the graphs are very yellow, the reported RMSPBE was either inf or became numerically unstable and thus no valid data was attainable. Readers may question why there are some parameters for which our methods are divergent: while our gradient-TD methods are guaranteed to converge, this is not for all parameter settings. In particular, the convergence guarantee for gradient-TD puts several constraints on α and β which are violated by our use of fixed valued α and β parameters. Thus it is not unreasonable that we see divergent behaviour with some of our parameter settings.

is thus a major caveat to GQ+'s success. Ideally, we would like an algorithm where little knowledge about the problem is needed to set the algorithm's parameters. On this note, the next chapter explores GQ+'s performance in an off-policy version of this experiment to determine if GQ+'s parameter settings really are significantly different in off-policy problems in practice.

It is, however, not entirely fair to dismiss GQ and GQ+'s superior performance when compared to the other gradient-TD methods as a mere trick of parameter optimization. In figure 4.7 we look at the RMSPBE averaged over all episodes and runs for each value of η using the corresponding best α value. In this graph, we can see that even with larger values of η, even as large as $\eta = 2$, the performance of GQ beats GQ2 (and GQ-NEU is omitted from these graphs as the performance is significantly worse than the other algorithms and is difficult to display on the graphs without eclipsing the differences between our other algorithms). We conducted further experiments using GQ2 with larger values of η. The results are not shown here as GQ2's performance either plateaus or worsens with values of η larger than 4 on all three feature sets. We mention it, however, as it confirms that GQ2's performance is not the result of poor parameter selection. Even with larger values of η, GQoffers superior performance when compared to GQ2 on this domain.

4.6 Conclusion

In this simple on-policy experiment we are able to find support for the claim that gradient-TD methods are slower than conventional-TD methods on on-policy prediction problems. This claim is a large motivating factor for our goal to find faster gradient-TD methods in this thesis. Aside from the comparison between gradient- and conventional-TD methods, this experiment also gives us a couple of promising observations about our gradient-TD methods. First, amoung our original gradient-td algorithms, GQ offers consistently faster convergence than GQ2 and GQ-NEU in this problem. This trend motivates us to focus primarily on algorithms related to GQ for the remainder of this thesis. The second promising observation is that the modified algorithm GQ+ shows noticeable improvement on GQ. We cautiously note that despite good performance in this experiment, further experimentation is necessary to show how effective our recomputed TD-error trick is on larger and more difficult problems. We also hold reservations about our gradient-TD method's need for two learning parameters and how robust these algorithms are to the setting of these parameters. In particular, we wonder if optimal parameter settings for our

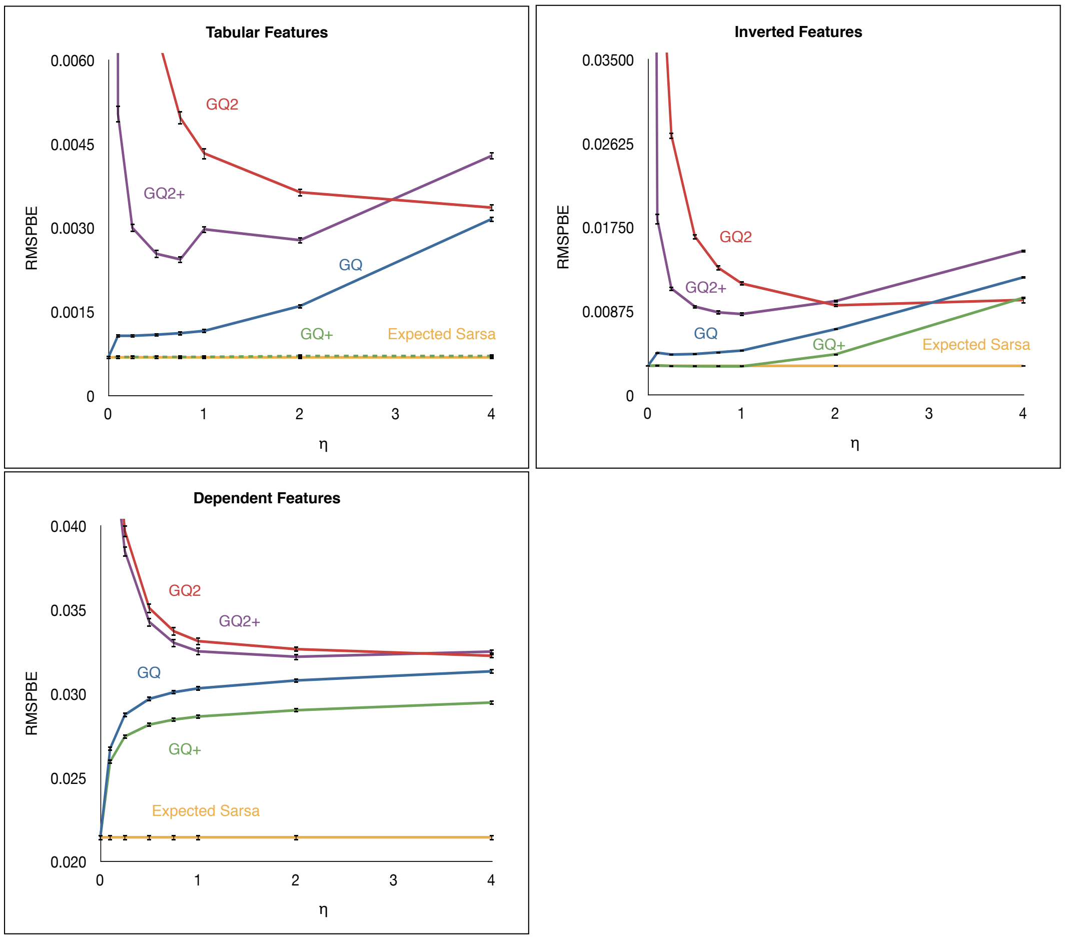

Figure 4.7: Graphing the RMSPBE of GQ and GQ2 and Expected Sarsa over various values of η. The best α for each η value is graphed here

gradient-TD methods will be greatly affected by switching to an off-policy problem and whether our recomputing TD-error trick will improve performance on off-policy problems as they have done on this on-policy problem. In the next chapter we will explore the performance of our gradient-TD methods on off-policy problems in order to examine these concerns further.

Chapter 5

Assessment on Off-Policy Prediction Problems

In this chapter we continue to explore the performance of GQ, GQ2, and GQ-NEU, testing their performance on off-policy problems based on the random-walk environment seen in the previous chapter. This experiment explores the question of how changes in behaviour and target policy affect the performance of our gradient-TD algorithms. We also explore whether the relative performance of our gradient-TD methods in an off-policy setting is consistent with our on-policy results. We will also take a look at the performance of our gradient-TD methods using the recomputed TD-error trick, to see if this modification still improves performance. In addition, in a final section of this chapter, we introduce two new GQ variants—Factored GQ and Double-Factored GQ–and present negative results showing that they may not be the faster gradient-TD we are looking for.

5.1 Algorithms

In this experiment we will evaluate the performance of our gradient-TD algorithms (both the originals, and the recomputed TD-error variants). To avoid unnecessary repetition, a detailed introduction to GQ, GQ2, and GQ-NEU appears in chapter 3, and an introduction to GQ+, GQ2+, and GQ-NEU+ can be found in chapter 4. We also include Expected Sarsa in our experiments. While it is not strictly fair to compare our gradient-TD methods to Expected Sarsa on off-policy problems, given Expected Sarsa is not guaranteed to converge, we continue to include Expected Sarsa in these experiments as an optimistic reference point of comparison. Expected Sarsa remains convergent on the problems used in this experiment and, as we will see, performs well. An introduction to the Expected Sarsa algorithm can be found in chapter 2, and a discussion of why Expected Sarsa is used to represent conventional-

TD algorithms in our experiments, can be found in the algorithms section of chapter 4.

5.2 Off-Policy Random Walk with Actions Problem

The experimental design for this chapter is very similar to the experiment presented in the previous chapter: GQ, GQ2, GQ-NEU, GQ+, GQ2+, GQ-NEU+, and Expected Sarsa algorithms are all tasked with learning a linear state-action value function for the random-walk environment used in the previous chapter. We again look at problems using three different state-action feature sets—however, in this experiment we look at a larger set of problems, now introducing a variety of possible behaviour and target policies. A description of the random walk environment and the three feature sets (tabular, inverted and dependent features) used in this experiment is provided in the problem section of the previous chapter.

As discussed, the purpose of this chapter is to evaluate the behaviour of the gradient-TD algorithms and Expected Sarsa on off-policy problems. Where in the previous chapter, our agents behaved with an equal 50-50 chance of moving left and right and learned the matching value function, in this experiment we now independently vary the behaviour and target policies across problems. In each unique problem, an agent now has a unique behaviour and target policy, both are kept static throughout the experiment.

Our policies always choose the right and left action with a fixed probability, regardless of the state they are in. For example, the behaviour policy used in the previous chapter always chooses the right action 50% of the time, no matter which state the agent is in. Because the policies can be summed up by their probability of moving right, we can use the shorthand notation B50 to refer to the behaviour policy that moves right 50% of the time. Similarly, T20 would be the target policy that moves right 20% of the time.

As the behaviour and target policies become more different, an agent spends more time learning about actions which may not be taken often by the target policy, and less time learning about taking the action the target policy cares about. In the most extreme case, the behaviour policy never chooses an action that the target policy would choose. In such a situation, it is impossible for the agent to learn the value of the action. We do not look at any such extreme cases. In our experiment, a problem is made for each possible pairing of the behaviour and target policies shown

behaviour policy	B20	B30	B40	B50	B60	B70	B80		
target policy	T10	T20	T30	T40	T50	T60	T70	T80	T90

Table 5.1: A table of the behaviour and target policies used in this experiment. Here B20 means the agent chooses the right action 20% of the time and chooses left the remaining 80%. An experiment set was run for every possible crosswise pairing of behaviour and target policy from this chart.

α	0.001	0.01	0.1	0.25	0.5	0.9	1.0	1.5	2.0
η	0.25	0.5	1	2	4				

Table 5.2: A table of agent parameters used where the gradient-td second learning rate β is a scalar multiple of α where $\beta = \alpha * \eta$. Each unique α and η pairing were tested.

in table 5.1. Note that most of the possible pairings are off-policy, however pairings like B20-T20 remain in the set, giving us results from on-policy problems with more varied behaviour policies than in our previous chapter's experiment.

5.3 Experimental Settings

As in the previous chapter, to fully specify our experimental settings, we must look at the algorithm parameter settings. Recall that Expected Sarsa has one learning rate parameter α, where all the gradient-TD algorithms have both α and β. As was discussed in the previous chapter's experiment section, an appropriate value of β is relative to the value of α being used. Therefore, to avoid having to provide multiple tables of β values, and to make the reporting of results easier, we use a variable η to calculate appropriate β for each α, where $\beta = \alpha * \eta$. Relying upon experience gleaned in the previous chapter, we are able to cull the set of α and η values used in our experiments—the newly trim list of values can be found in table 5.2. Expected Sarsa was tested on all α values, and the gradient-TD algorithms were tested on all possibly combinations of α and η.

The algorithms were run on the random walk environment using every possible behaviour and target policy pairing from table 5.1 for each of the three state-action feature sets. Each of these experiments ran for 50 runs of 500 time-steps each (again, after each run the agent's learned weights were reset). The reader may note that we are now running for a fixed number of time-steps, as opposed to the 1000 episodes long runs done in the on-policy experiment. This change to a fixed number of time-steps is necessary as a fixed number of episodes has a drastically different number of time-steps when using different behaviour policies. For example, if we compare a B50 policy, which chooses to go left or right with equal probability, to a B80 policy, which goes right 80% of the time and left the other 20% of the time, the expected number of time-steps per episode for the B50 policy is larger than that of the B80 policy. As such, to make sure each agent will see the same amount of data, these experiments are run for a fixed number of time-steps. Using only 500 time-steps may seem low given our previous experiment was run over 1000 episodes. However, 1000 episodes was vastly excessive as fewer than 200 episodes were needed in our previous experiment to get our algorithms to a steady point very close to convergence of 0.

After every time-step in our experiments, the root mean squared projected Bellman error (RMSPBE) of the agent was recorded. For a more detailed explanation of the RMSPBE, please see chapter 3, and for a discussion on why we use RMPSBE for our performance measure over other error measures, see chapter 4.

5.4 Learning Rate of Conventional- and Gradient-TD Methods on Off-Policy Problems

In analyzing the results of this experiment, we ask ourselves three questions:

1. How does varying the behaviour policy affect algorithmic performance?
2. How does varying the target policy affect algorithmic performance?
3. Is there, ultimately, a “fastest” algorithm?

To address these three questions, we present figures 5.1 and 5.2. In figure 5.1, we look at how the mean RMSPBE (the RMSPBE averaged over all runs and time-steps) is affected when the behaviour policy is kept constant, using B50, while the target policy is varied. Each algorithm uses a single pair of α and η values for all target policies; the value of α and η were chosen to minimize the mean RMSPBE averaged over all the target policies. We chose to keep α and η constant to give a better representation of each algorithm's general performance. We also looked at the results when the best parameters were selected for each target policy and found that the trends amongst the data were unchanging. Similarly, figure 5.2 shows how the mean RMSPBE is affected when the target policy is kept constant, using T50, while the behaviour policy is varied. Again, we used the average best parameter settings, which performed best on average over the behaviour policies.

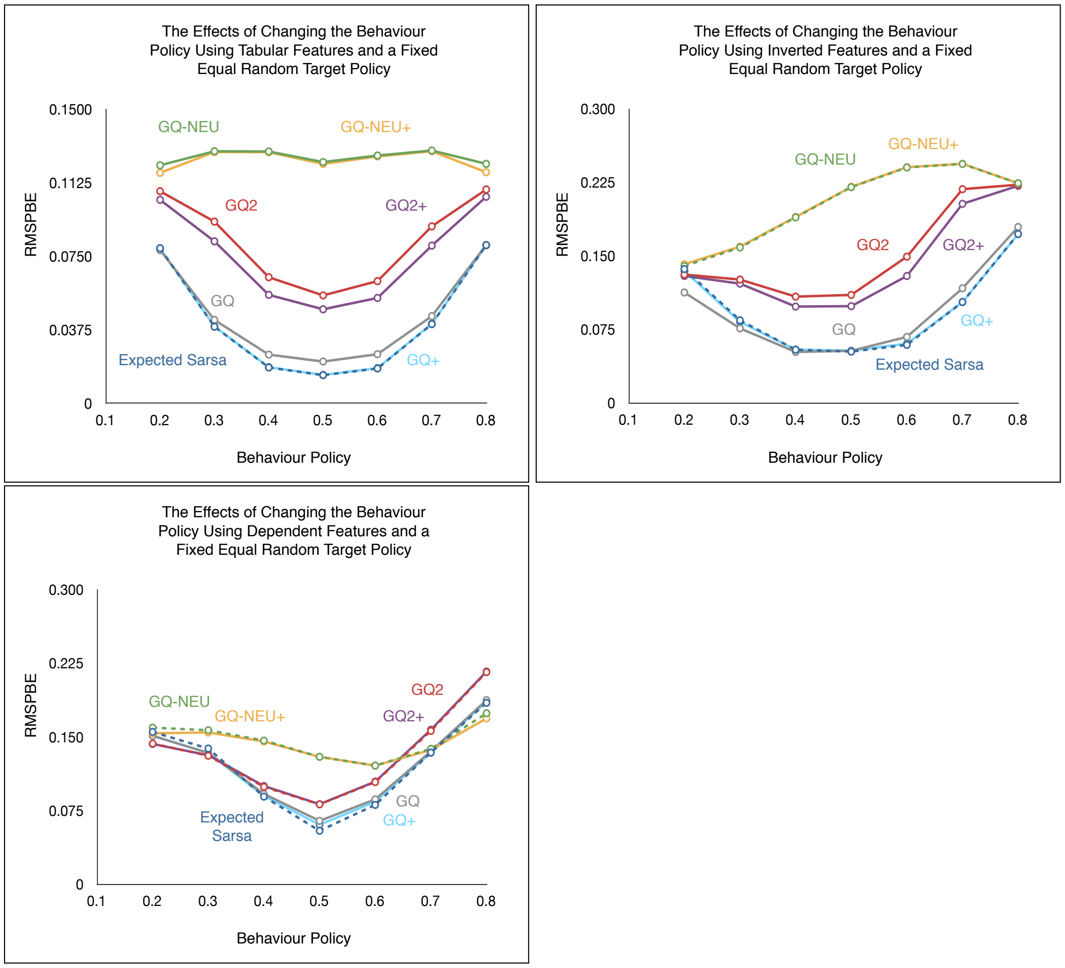

Figure 5.1: Using a fixed behaviour policy that chooses the actions left and right with equal probability, these graphs show how modifying the target policy (indicated as a probability of selecting the right action) affects the RMSPBE averaged over all 50 runs and 500 time-steps. The values for α and η used by each algorithm are the values which have the minimum mean RMSPBE averaged over all target polices. We omit some of the data points for GQ-NEU and GQ-NEU+ on the dependent features problem as their performance was significantly worse than the other algorithms and made presentation of all the algorithms more difficult. Standard error bars are left off the graph as the errors are smaller than the symbols used to mark data points.

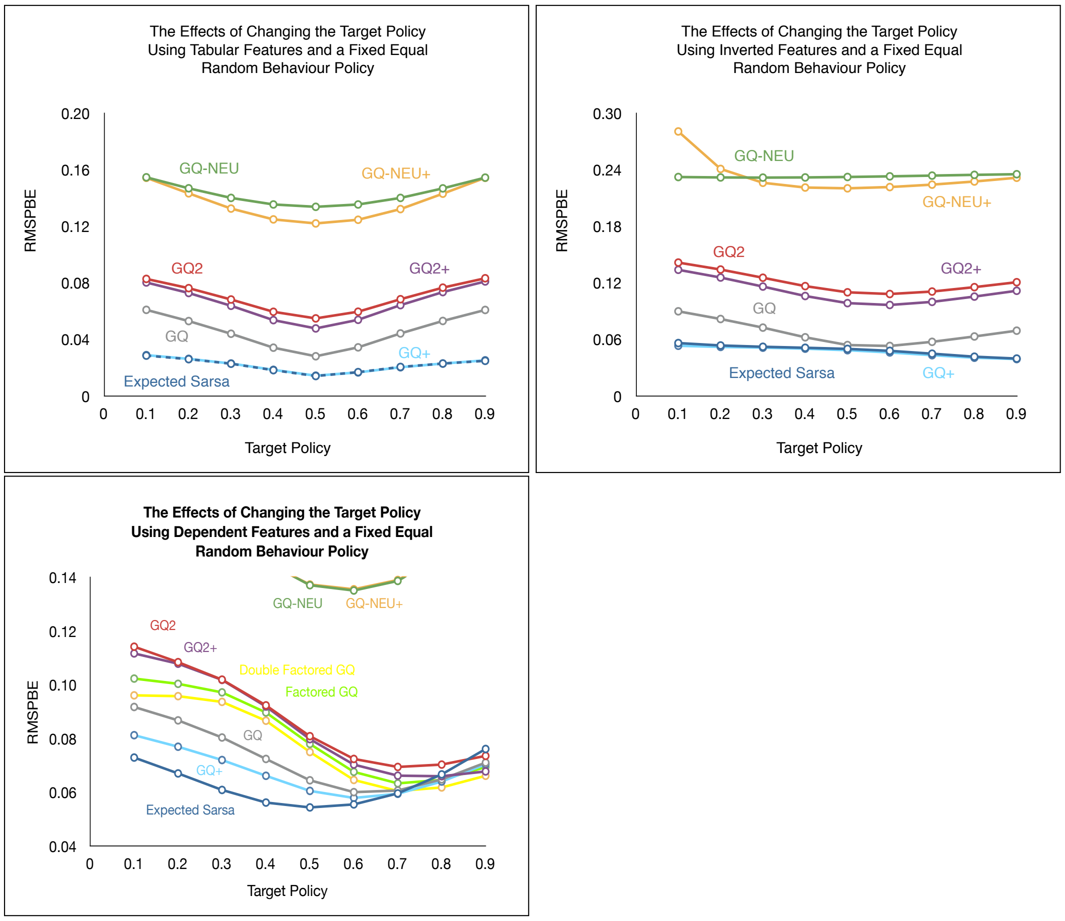

Figure 5.2: Using a fixed target policy that chooses the actions left and right with equal probability, these graphs show how modifying the behaviour policy affects the RMSPBE averaged over all 50 runs and 500 time-steps. The values for α and η are the values which have the minimum mean RMSPBE averaged over all behaviour polices. Standard error bars are left off the graph as the errors are smaller than the symbols used to mark data points.

Before comparing the relative performance of the various algorithms, we need to comment on several algorithmic independent trends in the data. One trend amongst these two figures shows that varying the behaviour policy has a much more drastic effect on the performance of our algorithms than the target policy. This is perhaps unsurprising, as the unavailability of data can be an extremely limiting factor—if a behaviour policy does not take certain actions frequently, the agent is severely limited in how well it can learn the value of those actions. Conversely, if the target policy does not take certain actions, but the behaviour policy does, the agent's ability to learn is only hindered in so far that it has extra unnecessary data.

While changes in the behaviour policy seems to have the strongest effect on algorithmic performance, the distance between our target and behaviour policies unsurprisingly relates to the performance as well. As the target and behaviour policy become more divergent, the performance of all the algorithms goes down. This trend was previously observed by Delp in his master's thesis (Delp 2011).

There is an asymmetry in our results that may cause some readers to pause. In figure 5.1, we see that performance on problems with target policies that choose the left action more frequently have poorer results than those problems with right-action dominant target policies. Similarly, figure 5.2 has an asymmetry where problems with behaviour policies that favour moving right seem to have poorer results than problems with behaviour policies that are left action dominant. It seems important to point these asymmetries out as a likely unanticipated artifact of how the value functions for each feature representation is initialized. Recall that the agent is given a reward of 1 for reaching the right most terminal state, and a reward of 0 for reaching the left most terminal state. If the initial value estimate for all states is 0, then this value function is already closer to the true value function for problems where the target policy moves left more frequently. An optimistic value function is therefore closer to the true value function for problems with target policies that move right more frequently.

Using $\theta(i) = 0.5, \forall i$ as the initial weight vector gives a positive estimate for the value of all state-action pairs for all three feature functions—the tabular case starts with an initial value of 0.5 for all state-action pairs, the inverted case starts with an initial value of 1.0 for all state-action pairs, and the dependent case has initial values of $\hat{Q}(s_0, a_{left}) = \hat{Q}(s_0, a_{right}) = 0.5, \hat{Q}(s_1, a_{left}) = \hat{Q}(s_1, a_{right}) = 0.70711, \hat{Q}(s_2, a_{left}) = \hat{Q}(s_2, a_{right}) = 0.86603, \hat{Q}(s_3, a_{left}) = \hat{Q}(s_3, a_{right}) = 0.70711, \hat{Q}(s_4, a_{left}) = \hat{Q}(s_4, a_{right}) = 0.5$. If the target policy is heavily biased to move left, these initial value functions are significantly optimistic, where as they would be closer to the true value function for target policies which favour the right most action. This bias in our initial value function thus explains the asymmetry seen in figure 5.1. Similarly, when the behaviour policy is biased to move right, the agent will see more samples with the terminal reward of 1, which means it will take longer to deflate an overly optimistic initial value function which explains the trend in figure 5.2 for agents to take longer to solve the problems with behaviour policies that are biased towards moving right. As such, the asymmetries in figures 5.1 and 5.2 show us the potential effects of selecting good and bad initial value functions.

Regardless of these effects, there is however a clear, consistent trend in the relative performances of all the gradient-TD algorithms. Similar to the on-policy results from the previous chapter, we see that GQ is the clear winner over GQ-NEU and GQ2. We also see that the recomputed TD-error algorithms continue to be provide an improvement in performance, however the effect is again most noticeable in the case of GQ+.

One lingering question from the previous chapter is whether GQ and GQ+ will continue to be robust to parameter setting in the off-policy setting.There was concern that GQ and GQ+'s performance was overly dependent on the value of η chosen and that the range of acceptable values for this algorithm were narrow. In particular, there was concern that the optimal parameter values of GQ and GQ+ may be highly problem dependent—would changing our problem to be off-policy affect the range of well-performing parameter values? In the context of this experiment, the answer appears to be no: when looking at results using a given feature set, the optimal parameter values for GQ and GQ+ rarely change, and when they do, the difference in performance is slight. This seems to suggest that while it is still necessary to find good parameters, that these parameters are not greatly affected by changing our policies, which is a desirable quality to have in the more difficult control problem where the behaviour policy may be changing as the agent learns. This is also desirable in situations where the agent is learning about multiple target policies at once from the same data—one would not want to have to find the optimal parameters for each target policy.

5.5 Factored GQ and Double-Factored GQ

Factored GQ and Double-Factored GQ are two new variations of GQ that we developed as hopeful contenders for the title of faster GQ. These algorithms take their name from the fact that they are variations of GQ where we have factored the gradient used by GQ into smaller terms, allowing us to use samples to approximate a larger portion of the gradient. Recall that GQ uses the following expression of the gradient of the mean squared projected Bellman error:

$$-\frac{1}{2}\nabla MSPBE(\theta) = \mathbb{E}\left[\delta_t\phi_t\right] - \gamma\,\mathbb{E}\left[\bar{\phi}^{\pi}_{t+1}\phi_t^T\right]\mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[\delta_t\phi_t\right].$$

Unfortunately, because the GQ gradient involves multiplying several expectations with shared random variables, this gradient cannot be sampled in its entirety without inducing bias. As such, GQ only uses samples to approximate a portion of the gradient. The remaining portion of the gradient is approximated by a learned set of linear weights. GQ replaces the term $\mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[\delta_t\phi_t\right]$ in the gradient with the learned weights w:

$$-\frac{1}{2}\nabla MSPBE(\theta) = \mathbb{E}\left[\delta_t\phi_t\right] - \gamma\,\mathbb{E}\left[\bar{\phi}^{\pi}_{t+1}\phi_t^T\right] w.$$

This allows GQ to use samples in place of the rest of the gradient. Factored GQ has the same derivation of GQ up until this point where the gradient is divided into sampled and estimated segments. At this point, Factored GQ takes the portion of the GQ gradient estimated by w and factors it into smaller terms—some of which can be sampled without worry of inducing bias into our gradient. To begin, we replace δ with its expanded form:

$$\begin{aligned} w &= \mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[\delta_t\phi_t\right] \\ &= \mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[(r_t + \gamma\theta_t^T\bar{\phi}^{\pi}_{t+1} - \theta_t^T\phi_t)\phi_t\right] \end{aligned}$$

Once we replace δ with its long form, a small amount of algebraic manipulation allows us to isolate a lone $\mathbb{E}\left[\theta\right]$ term.

$$\begin{aligned} w &= \mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[(r_t + \gamma\theta_t^T\bar{\phi}^{\pi}_{t+1} - \theta_t^T\phi_t)\phi_t\right] \\ &= \mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[r_t\phi_t + (\gamma\theta^T\bar{\phi}^{\pi}_{t+1})\phi_t - (\theta_t^T\phi_t)\phi_t\right] \\ &= \mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\left(\mathbb{E}\left[r_t\phi_t\right] + \mathbb{E}\left[\gamma\theta_t^T\bar{\phi}^{\pi}_{t+1})\phi_t\right] - \mathbb{E}\left[(\theta_t^T\phi_t)\phi_t\right]\right) \\ &= \mathbb{E}\left[\phi\phi^T\right]^{-1}\mathbb{E}\left[r\phi\right] + \mathbb{E}\left[\phi\phi^T\right]^{-1}\mathbb{E}\left[\gamma\theta^T\bar{\phi}^{\pi}_{t+1})\phi\right] - \mathbb{E}\left[\phi\phi^T\right]^{-1}\mathbb{E}\left[\phi(\phi^T\theta)\right] \\ &= \mathbb{E}\left[\phi\phi^T\right]^{-1}\mathbb{E}\left[r\phi\right] + \mathbb{E}\left[\phi\phi^T\right]^{-1}\mathbb{E}\left[\gamma\theta^T\bar{\phi}^{\pi}_{t+1})\phi\right] - \mathbb{E}\left[\theta\right] \end{aligned}$$

We can sample this θ term without worrying about inducing bias to our gradient estimate and so Factored GQ can sample a larger part of the gradient and learn a set of weights $\tilde{w}$, just like GQ learns w:

$$\begin{aligned}
\tilde{w} &= w - \mathbb{E}\left[\theta_t\right] \\
\tilde{w} &= \mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[r_t\phi_t\right] + \mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[\gamma\theta_t^T\bar{\phi}_{t+1}^{\pi})\phi_t\right]
\end{aligned}$$

The learning update for Factored GQ is as follows:

$$\begin{aligned}
\delta_t &= r_t - \gamma\theta^T\bar{\phi}_t^{\pi} + \theta^T\phi_t \\
\theta_{t+1} &= \theta_t + \alpha(\delta\phi - \bar{\phi}_t^{pi}(\phi^T(\tilde{w}_t - \theta_t))) \\
\tilde{w}_{t+1} &= \tilde{w}_t + \beta\left[(r + \theta^T\bar{\phi}_t^{\pi})\phi - (\tilde{w}_t^T\phi_t)\phi_t\right] \qquad (5.1)
\end{aligned}$$

Double-Factored GQ applies a very minor change to Factored GQ: Double-Factored GQ breaks up $\tilde{w}$ into two separate learnable weight vectors $\dot{w} = \mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[(r_t\phi_t\right]$ and $\dot{v} = \mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[(\theta_t^T\bar{\phi}_{t+1}^{\pi})\phi_t\right]$. The gradient written using $\dot{w}$ and $\dot{v}$ is as follows:

$$\begin{aligned}
-\frac{1}{2}MSPBE(\theta) &= \mathbb{E}\left[\delta_t\phi_t\right] - \gamma\,\mathbb{E}\left[\bar{\phi}_{t+1}^{\pi}\phi_t^T\right]\left[\mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[r_t\phi_t\right] + \gamma\,\mathbb{E}\left[\phi_t\phi_t^T\right]^{-1}\mathbb{E}\left[(\theta_t^T\bar{\phi}_{t+1}^{\pi})\phi_t\right] - \mathbb{E}\left[\theta_t\right]\right] \\
&= \mathbb{E}\left[\delta_t\phi_t\right] - \gamma\,\mathbb{E}\left[\bar{\phi}_{t+1}^{\pi}\phi_t^T\right]\left[\dot{w} + \dot{v} - \mathbb{E}\left[\theta_t\right]\right]
\end{aligned}$$

The corresponding updates for the Double-Factored GQalgorithm are:

$$\begin{aligned}
\delta_t &= r_t - \gamma\theta^T\bar{\phi}_{t\pi} + \theta^T\phi_t \\
\theta_{t+1} &= \theta_t + \alpha(\delta\phi - \bar{\phi}_{tpi}(\phi^T(\dot{w}_t + \dot{v}_t - \theta_t))) \\
\dot{w}_{t+1} &= \dot{w}_t + \beta_{\dot{w}}\left[r\phi - (\dot{w}_t^T\phi_t)\phi_t\right] \\
\dot{v}_{t+1} &= \dot{v}_t + \beta_{\dot{v}}\left[(\theta\bar{\phi}_t^{\pi})\phi - (\dot{v}_t^T\phi_t)\phi_t\right]
\end{aligned}$$

At this point, the reader should note that when $\beta_{\dot{w}}$ and $\beta_{\dot{v}}$ are equal, Double-Factored GQ is the same algorithm as Factored GQ. As Double-Factored GQ introduces yet a third free parameter, we choose not to focus heavily on Double-Factored GQ in this thesis. We include Double-Factored GQ in the following experiment in a very limited capacity—setting $\beta_{\dot{w}}$ to be half of $\beta_{\dot{v}}$ for all experiments—to show that Double-Factored GQ can have different performance from Factored GQ. However, we leave a thorough study of this difference to future work.

5.6 Learning Rate of Factored GQ and Double-Factored GQ on Off-policy Problems

We ran Factored GQ and Double-Factored GQ on the same experiment as above, running our algorithms for 50 runs of 500 time-steps on all of our problems varying the target and behaviour policies. We used the same α and η values—using $\beta = \alpha * \eta$ for Factored GQ and $\beta_{\dot{v}} = \alpha * \eta$ and $\beta_{\dot{w}} = \frac{1}{2}\beta_{\dot{v}}$ for Double-Factored GQ. Similar to our previous experiment, we graph how the averaged RMSPBE of Factored GQ and Double-Factored GQ is affected when the behaviour policy is held fixed, and the target policy is changed in figure 5.3, and how things change when the target policy is held fixed and the behaviour policy is changed in figure 5.4. The average best parameters were used in each graph, meaning that α and η were the parameters which had the lowest average RMSPBE when averaged over all the behaviour policies in the case of figure 5.3, and all the target policies in the case of figure 5.4.

In general, these graphs show that Factored GQ and Double-Factored GQ perform worse than GQ. Of course, this experiment did not look at the effects of varying the third step-size parameter $\beta_{\dot{w}}$ for the Double-Factored GQ algorithm and so further study is necessary to truly evaluate Double-Factored GQ's potential. However, from these limited results, we do not believe the addition of the third step-size parameter can yield improvements significant enough to compensate for the extra complexity added by the need to optimize a third parameter.

5.7 Conclusions

In this chapter, we have seen the continued dominance of GQ and GQ+ over our other gradient-TD methods. While the performance of GQ+ is mostly on par with Expected Sarsa in these experiments, the results on the dependent feature problems shown in figure 5.1 show some that there are cases where GQ+ is still slower than Expected Sarsa. As the dependent feature set is arguably a difficult feature set to

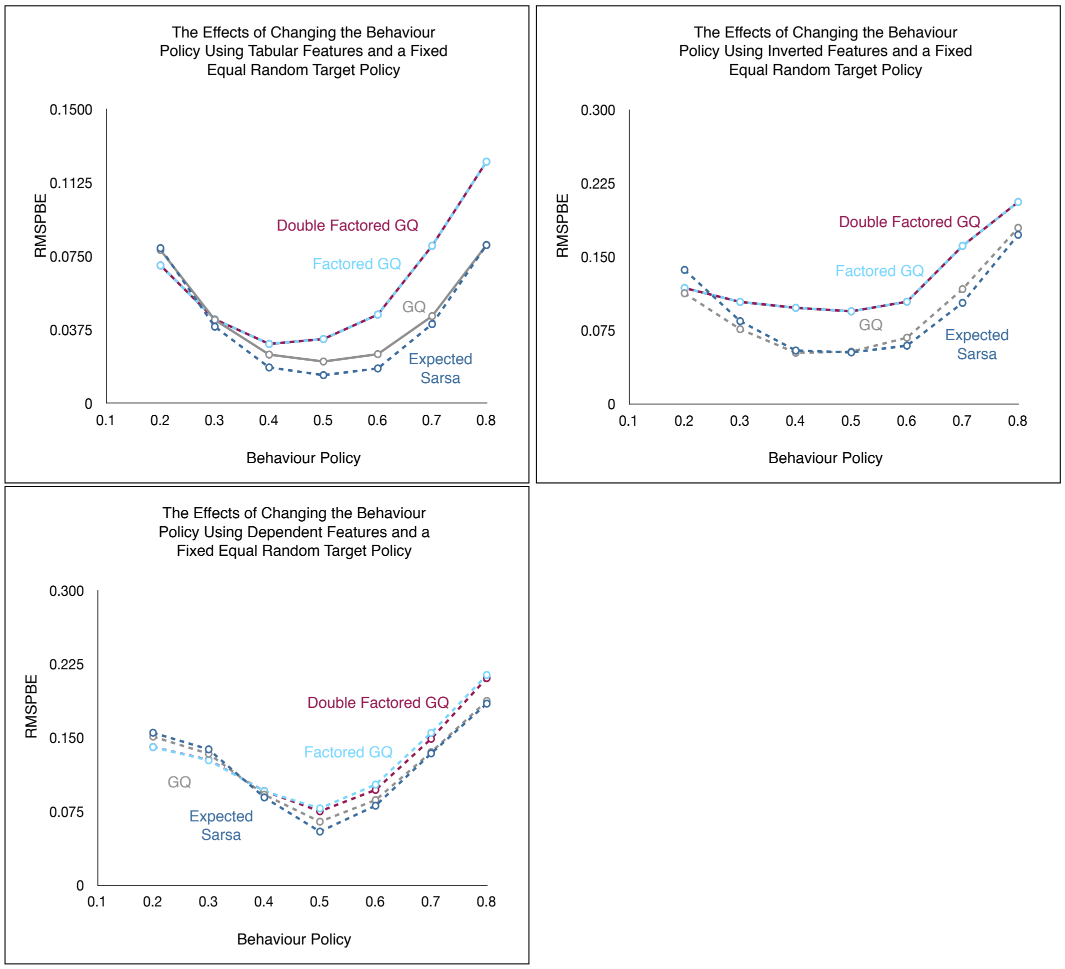

Figure 5.3: Using a fixed behaviour policy that chooses the actions left and right with equal probability, these graphs show how modifying the target policy (indicated as a probability of selecting the right action) affects the RMSPBE averaged over all 50 runs and 500 time-steps. The values for α and η used by each algorithm are the values which have the minimum mean RMSPBE averaged over all target polices. Standard error bars are left off the graph as the errors are smaller than the symbols used to mark data points.

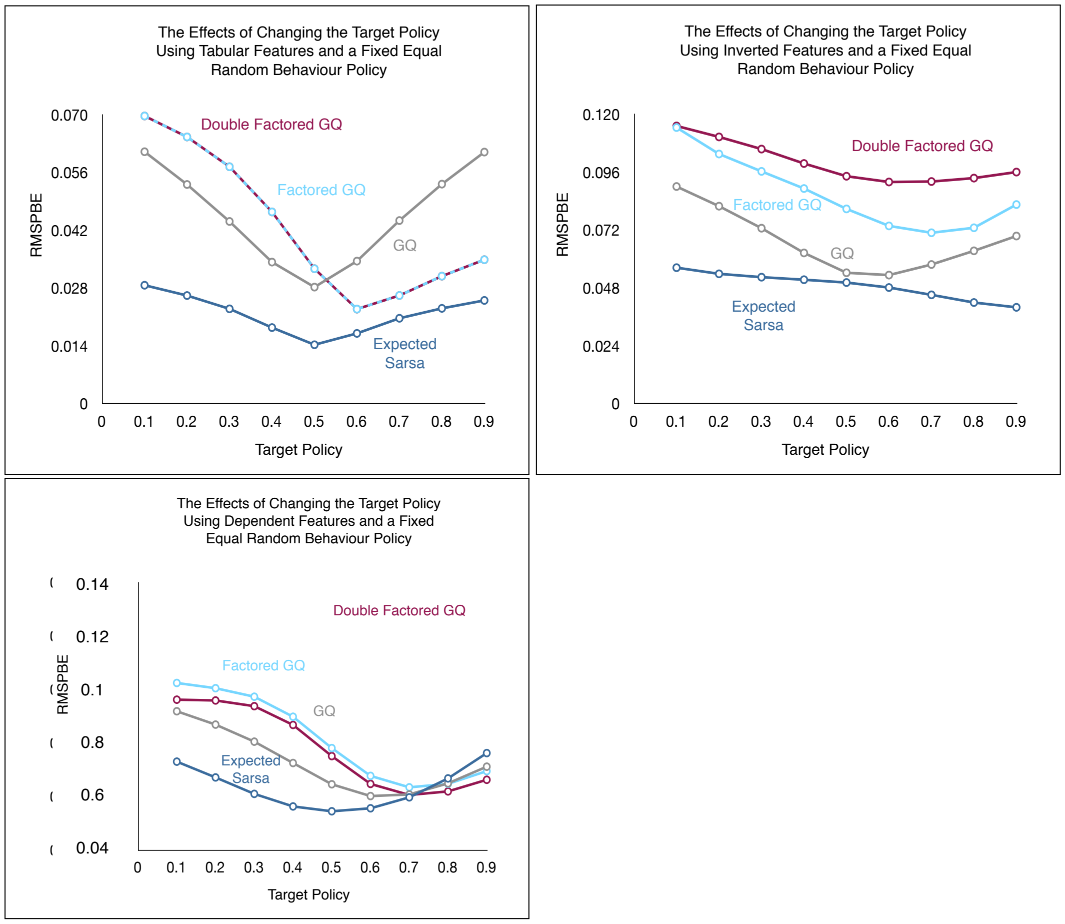

Figure 5.4: Using a fixed target policy that chooses the actions left and right with equal probability, these graphs show how modifying the target policy (indicated as a probability of selecting the right action) affects the RMSPBE averaged over all 50 runs and 500 time-steps. The values for α and η used by each algorithm are the values which have the minimum mean RMSPBE averaged over all behaviour polices. Standard error bars are left off the graph as the errors are smaller than the symbols used to mark data points.

use for this problem, this may suggest that on harder problems, there may be a larger gap between Expected Sarsa and GQ+. This suggests that future work on more difficult problems is needed.

Lastly, we note that our factored methods, Factored GQ and Double-Factored GQ, did not provide a faster gradient-TD method as we had hoped. With this in mind, we continue in the next chapter in our search for faster gradient-TD methods by introducing a set of new hybrid gradient-TD algorithms.

Chapter 6

Hybrid Gradient-TD Methods

In chapter 4, we saw that the performance of Expected Sarsa seemed to act as a lower bound for the performance of our gradient-TD agents in on-policy problems and while it was possible to obtain equitable performance under favourable conditions, we were unable to beat Expected Sarsa on on-policy problems. From this observation, an idea was born: what we would like are algorithms whose update on on-policy problems is the same as Expected Sarsa (or other fast on-policy algorithms) and then, on off-policy problems, the update changes to be more like GQ, to ensure the algorithm converges on off-policy problems. We have already seen that GQ has the same update as Expected Sarsa when w and β are set to 0, however we desire an algorithm that behaves like Expected Sarsa on on-policy problems automatically, without intervention on the part of the experimentor. We refer to algorithms that have this ability—to switch between on-policy updates and off-policy updates automatically when necessarily, without external intervention—as hybrid gradient-TD algorithms.

This chapter introduces three new algorithms, two of which are hybrid gradient-TD algorithms. Our new algorithms are named AB, Hybrid-GQ, and TDGQ and were developed by Hamid Maei and Rich Sutton. At time of publication, AB and Hybrid-GQ appear only in a brief discussion in the appendix of (Maei 2011), and TDGQ has never before appeared in publication. No experimental results exist with any of these three algorithms. In this chapter, we will provide a more formal presentation of these algorithms, and provide some intuition to why these algorithms are convergent and may be effective. The next chapter then presents initial experimental results using these new algorithms.

6.1 AB and Hybrid-GQ

In this section we present two new gradient-TD algorithms: AB and Hybrid-GQ. While we began this chapter discussing our interest in creating hybrid gradient-TD methods, the development of AB and Hybrid-GQ is initially not related to hybrid gradient-TD methods: these two algorithms find their beginning in a simple idea for modifying GQ to find new convergent gradient-TD methods, an idea which is unrelated to hybrid gradient-TD methods. It is only a happy coincidence that Hybrid-GQ is a hybrid gradient-TD algorithm.

The inspiration and development of AB and Hybrid-GQ, can best be summarized as an exercise in cleverness, optimism and luck. The initial cleverness lies in two observations about the gradient of the mean-square projected Bellman error (MSPBE). For the purpose of this discussion, let us rewrite the MSPBE in a different form than usual:

$$\begin{aligned}\text{MSPBE}(\theta) &= \mathbb{E}\left[\delta_t\phi_t\right]^\top \mathbb{E}\left[\phi_t\phi_t^\top\right]^{-1}\mathbb{E}\left[\delta_t\phi_t\right]\\ &= (-A_\pi\theta + b)^\top C^{-1}(-A_\pi\theta + b)\end{aligned}$$

Where $A_\pi = \mathbb{E}\left[(\phi_t - \gamma\bar{\phi}^\pi_{t+1})\phi^\top\right]$, $b = \mathbb{E}\left[r_t\phi_t\right]$, and $C = \mathbb{E}\left[\phi_t\phi_t^\top\right]$. Note we include the subscript π on A_π to indicate that the next state, $\bar{\phi}^\pi_{t+1}$ is drawn according to the target policy π. The MSPBE gradient, written in terms of these matrices is then:

$$\begin{aligned}-\frac{1}{2}\nabla MSPBE(\theta) &= \mathbb{E}\left[(\phi_t - \gamma\bar{\phi}^\pi_{t+1})\phi_t^\top\right]\mathbb{E}\left[\phi_t\phi_t^\top\right]^{-1}\mathbb{E}\left[\delta_t\phi_t\right]\\ &= A_\pi^\top C^{-1}(-A_\pi\theta + b),\end{aligned}$$

The first key observation that we must make here is to first realize that to ensure convergence, $A_\pi^\top C^{-1}A_\pi$ must be positive semi definite, and to ensure that this is true, we only need guarantee that C^{-1} is positive semi definite (Maei & Sutton 2010). The second key observation to make is that the fixed-point of the MSPBE is independent of C^{-1}. Accordingly, we could replace C^{-1} with any semi positive definite matrix that is independent of θ and our algorithm will still be guaranteed to converge to the same fixed-point. Note however that while replacing C^{-1} does not affect the fixed-point, it does affect the rate of convergence. Recall that the norm of the expected update (NEU) (see equation 3.2), the objective function of GQ-NEU, is equivalent to replacing C^{-1} with the identity matrix. As we have seen in our experiments with GQ-NEU and GQ, this difference is enough that GQ-NEU converges significantly slower than GQ.

This is where optimism comes into the development of AB and Hybrid-GQ: the idea behind both algorithms is to find a matrix to replace C^{-1} in the expected update which will improve on GQ's rate of convergence. Note we are talking about substituting C in the gradient (the update for θ, $\Delta\theta$), thus the new update produced by changing C is not necessarily the gradient of anything. Thus, while any new update produced by replacing C is guaranteed to converge to the same value of θ, it is no longer a gradient descent method.

AB and Hybrid-GQ are the bi-product of replacing C^{-1} with matrices A_μ^{-1} and $A_\mu^{-1\top}$ respectively (where A_μ is the same as A_π except that ϕ' is now ϕ_{t+1}, drawn according to the behaviour policy μ). Note, we cannot use the matrix A_π itself, as A_π is not necessarily positive definite (Tsitsiklis & Van Roy, 1997). A_μ is positive definite because ϕ_t and $\bar{\phi}^\mu_{t+1}$ are both drawn according to the behaviour policy (Sutton 1988).

Looking at AB first, we being by replacing C^{-1} with A_μ^{-1} to create a modified gradient $A_\pi{}^\top A_\mu^{-1}(-A_\pi\theta + b)$. We then set about rearranging this modified gradient so that it has two distinct terms: one term which is the standard TD-style update and a second that is a correction term, just as in the update for GQ. We refer to updates which have a TD-update term, and a correction term as *modular updates.*

Using the knowledge that $A_\pi = C - \mathbb{E}\left[\gamma\phi_t\bar{\phi}^\pi_{t+1}{}^\top\right]$ and $A_\mu = C - \mathbb{E}\left[\gamma\phi_t\bar{\phi}^\mu_{t+1}{}^\top\right]$, we can rearrange this modified gradient as follows:

$$\begin{aligned}
A_\pi^\top A_\mu^{-1}(-A_\pi\theta + b) &= (A_\mu - A_\mu + A_\pi^\top)A_\mu^{-1}(-A_\pi\theta + b) \\
&= (-A_\pi\theta + b) + (-A_\mu + A_\pi^\top)A_\mu^{-1}(-A_\pi\theta + b) \\
&= \mathbb{E}\left[\delta\phi\right] - \gamma(-C + \mathbb{E}\left[\phi_t\bar{\phi}^\mu_{t+1}{}^\top\right] + C - \mathbb{E}\left[\bar{\phi}^\pi_{t+1}\phi_t^\top\right])A_\mu^{-1}(-A_\pi\theta + b) \\
&= \mathbb{E}\left[\delta\phi\right] - \gamma(\mathbb{E}\left[\bar{\phi}^\pi_{t+1}\phi_t^\top\right] - \mathbb{E}\left[\phi_t\bar{\phi}^\mu_{t+1}{}^\top\right])A_\mu^{-1}(-A_\pi\theta + b) \qquad (6.1)
\end{aligned}$$

Here it is important to pay careful attention to which expectations use the target policy distribution and which use the behaviour policy distribution. By default, all the previously discussed gradient-TD algorithms have been excursion methods, and as such all expectations have used the behaviour distribution for selecting ϕ_t and the target distribution for selecting the next state-action pair ($\bar{\phi}^{\pi}_{t+1}$). In these new algorithms, we are explicitly adding A_μ into the equation, which means that both $\bar{\phi}^{\mu}_{t+1}$ and $\bar{\phi}^{\pi}_{t+1}$ will be present.

From here, AB follows a similar pattern as our other gradient-TD methods. First, we introduce a linear weight vector w that approximates part of the gradient to avoid the bias introduced by taking the product of related expectations. For the algorithm AB, $w = A_\mu^{-1}(-A_\pi\theta + b)$. Then we can define a stochastic gradient descent style update for θ and an update for w.

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha_t\left[\delta_t\phi_t - \gamma\left(\bar{\phi}^{\pi}_{t+1}\phi_t^\top w - \phi_t\bar{\phi}^{\mu}_{t+1}w_t\right)\right] &(6.2)\\ w_{t+1} &= w_t + \beta_t\left(\delta_t + \gamma\, w_t^\top\bar{\phi}^{\mu}_{t+1} - w_t^\top\phi_t)\right)\phi_t &(6.3)\end{aligned}$$

To understand where the update for w comes from, notice that $w = A_\mu^{-1}(-A_\pi\theta + b)$ resembles the analytic TD solution using $r = \delta$. As such, we can learn w using a TD-style update using δ in lieu of the reward.

Here we see that while AB is not a hybrid gradient-TD method, it is in fact, very close: when μ and π are the same, if $\bar{\phi}^{\pi}_{t+1}\phi_t^\top$ was symmetric, the correction term would be zero. With the thought that we are but a mere transpose away from having a hybrid gradient-TD algorithm, we switch to the derivation of Hybrid-GQ. Again, we start by finding a form of the modified gradient $A_\pi^\top A_\mu^{\top^{-1}}(-A_\pi\theta + b)$ which will give us a modular update:

$$\begin{aligned}A_\pi^\top A_\mu^{\top^{-1}}(-A_\pi\theta + b) &= (A_\mu^\top - A_\mu^\top + A_\pi^\top)A_\mu^{\top^{-1}}(-A_\pi\theta + b)\\ &= (-A_\pi\theta + b) + (-A_\mu^\top + A_\pi^\top)A_\mu^{\top^{-1}}(-A_\pi\theta + b)\\ &= \mathbb{E}\left[\delta\phi\right] - \gamma(-C + \mathbb{E}\left[\phi_t\bar{\phi}^{\mu}_{t+1}{}^\top\right] + C - \mathbb{E}\left[\phi_t\bar{\phi}^{\pi}_{t+1}{}^\top\right])A_\mu^{\top^{-1}}(-A_\pi\theta + b)\\ &= \mathbb{E}\left[\delta\phi\right] - \gamma(\mathbb{E}\left[\bar{\phi}^{\pi}_{t+1}\phi_t^\top\right] - \mathbb{E}\left[\phi^{\mu}_{t+1}\phi_t^\top\right])A_\mu^{\top^{-1}}(-A_\pi\theta + b), &(6.4)\end{aligned}$$

From here, we can introduce a learned weight $w = A_\mu^{\top^{-1}}(-A_\pi\theta + b)$ and create a gradient-descent style update for θ, and update w as follows:

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha_t \left[\delta_t \phi_t - \gamma(\bar{\phi}^{\pi}_{t+1} - \bar{\phi}^{\mu}_{t+1})\phi_t^\top w_t\right] \\ w_{t+1} &= w_t + \beta_t \left[\left(\delta_t - \phi_t^\top w_t\right)\phi_t + \gamma(\phi_t^\top w_t)\bar{\phi}^{\mu}_{t+1}\right]. \end{aligned} \tag{6.5}$$

From the update of θ, it is clear that when μ and π are the same distributions, that $(\bar{\phi}'^{\pi} - \bar{\phi}^{\mu}_{t+1}) = 0$ and thus the correction term disappears and the update becomes a TD-update. As the two policies become increasingly divergent, the difference between $(\bar{\phi}'^{\pi} - \bar{\phi}^{\mu}_{t+1})$ becomes larger, meaning the correction term has a larger affect on the update. This is exactly the kind of behaviour we used to define hybrid gradient-TD algorithms. As such, by a stroke of luck, Hybrid-GQ is our first hybrid gradient-TD algorithm.

6.2 TDGQ

There is no simple motivating story to explain the creation of TDGQ: the algorithm was born from the desire to create a hybrid gradient-TD method. Instead of trying to slowly build TDGQ, we will present the algorithm in whole and then dissect it to provide some intuition on how this algorithm works, and finally sketch a proof showing that TDGQ converges to the TD fixed-point.

The update for TDGQ is as follows:

$$\theta_{t+1} = \theta_t + \alpha \left[r_{t+1} + \gamma \theta_{t+1}^{GQ\ \top}(\bar{\phi_{t+1}}^{\pi} - \bar{\phi}^{\mu}_{t+1}) + \gamma\theta_t^\top \bar{\phi}^{\mu}_{t+1} - \theta_t^\top \phi_t\right]\phi_t \tag{6.6}$$

where θ^{GQ} is the weight vector θ learned by GQ on the same data. We can see intuitively that this algorithm may be a hybrid gradient-TD method, as when our target and behaviour policy are the same, $(\bar{\phi}^{\pi}_{t+1} - \bar{\phi}^{\mu}_{t+1}) = 0$, and so our update becomes the Expected Sarsa update. What remains to be shown is that this method converges.

In essence, this algorithm is treating $r_{t+1} + \gamma\theta_{t+1}^{GQ\ \top}(\bar{\phi}^{\pi}_{t+1} - \bar{\phi}^{\mu}_{t+1})$ conceptually as a new reward signal. This new reward signal becomes increasingly different from the original reward signal as the updates become more off-policy. Treating $r_{t+1} + \gamma\theta_{t+1}^{GQ\ \top}(\bar{\phi}^{\pi}_{t+1} - \bar{\phi_{t+1}}^{\mu})$ as a new reward signal, we can see that after θ^{GQ} converges, our new modified reward signal becomes stable and the learning system becomes an ordinary on-policy Expected Sarsa system.

This means that θ is guaranteed to converge to a stable unique fixedpoint θ^{inf}. The question remains however, what does this algorithm converge to? Ideally, we would like our algorithm to converge to the TD fixpoint θ^*. If we can show that θ^* is a fixed point of TDGQ, this is sufficient to show that θ^{inf} must be θ^*, thus proving that TDGQ convertes to the TD fixed-point. First, let us note that if we rearrange the update for TDGQ] slightly, we can write our update as:

$$\theta_{t+1} = \theta_t + \alpha \left[r_{t+1} + \gamma(\theta_t - \theta_{t+1}^{GQ})^\top \bar{\phi}_{t+1}^{\mu} + \gamma \theta_{t+1}^{GQ\ \top} \bar{\phi}_{t+1}^{\pi} - \theta_t^\top \phi_t \right] \phi_t. \tag{6.7}$$

From here, if we can substitute θ^* for both θ_t and θ^{GQ} (given we know that θ^* is the fixed point of GQ), and show that this is a fixed point, we will have shown that $\theta^{\text{inf}} = \theta^*$. Once we substitute in θ^*, we get

$$\theta_{t+1} = \theta^* + \alpha \left[r_{t+1} + \gamma(\theta^* - \theta^*)^\top \bar{\phi}_{t+1}^{\mu} + \gamma \theta^{*\top} \bar{\phi}_{t+1}^{\pi} - \theta^{*\top} \phi_t \right] \phi_t. \tag{6.8}$$

which simplifies to be the update for Expected Sarsa. Given we know that Expect Sarsa's fixed-point is the TD fixed-point, then this update is at its fixed-point when $\theta = \theta^*$. Thus, θ^* is necessarily the fixed point for TDGQ.

We have now seen that the update for TDGQ becomes the Expected Sarsa update when encountering an on-policy update, and we've given a brief sketch for a proof that TDGQ will converge to the TD-fixed point, making TDGQ our second hybrid gradient-TD algorithm.

Chapter 7

Hybrid Algorithm Experiments

This final experiment chapter is a brief look at how our hybrid gradient-TD algorithms (Hybrid-GQ and TDGQ), and Hybrid-GQ's non-hybrid relative AB, compare to GQ and Expected Sarsa in the same off-policy problem as presented in chapter 5. We use the same experimental design as detailed in that off-policy experiment chapter, including the parameter values for α and η, the different behaviour and target policies evaluated, and the number of runs and time-steps for each experiment. To avoid repetitiveness, we point the reader to chapter 5 for a detailed description of the problem and instead focus only on presenting results here.

Note, that in this experiment, we do not compare our algorithms to GQ+, nor do we run variants of our hybrid algorithms that recompute the TD-error. This may surprise readers, given that GQ+ has continually outperformed GQ, and all of our gradient-TD algorithms have seen some degree of improvement from recomputing the TD-error. In truth, we did experiment with variants of the hybrid algorithms that recompute the TD-error between updates, however, the results showed that, with the exception of AB+, all our enhanced algorithms performed very similarly on this problem, suggesting that the problem may be too simple to show differences in their performance. We note that recomputing the TD-error seems to have a similarly beneficial effect on our hybrid algorithms. We leave such an assessment through further experimentation to future work.

Figures 7.1 and 7.2 are analagous to figures 5.1 and 5.2 presented in the off-policy experiment chapter. In these graphs, we look at the effect on performance of fixing our behaviour/target policy and then changing the target/behaviour policy. For a given algorithm on a given feature set, the values for α and η used were kept constant on each of the two figures: the parameter values selected for each of the two graphs were the average best parameters, showing the best performance when averaged across all behaviour/target policies explored in each graph. For example, in the case of figure 7.1, we select the parameters that had the lowest MSPBE averaged over all the target policies.

When the hybrid algorithms were introduced, the intuitional motivation behind these algorithms was that when a problem was on-policy, these algorithms would perform a conventional-TD update, and as a problem became increasingly off-policy, the algorithm's update would become more like a gradient-TD algorithm. In the case of Hybrid-GQ, we are able to express our update in terms of two separable terms: a traditional-TD update (the update for Expected Sarsa in the case of our two algorithms) and then a correction term. As the target and behaviour policies for a problem become more divergent, the weight put on the correction term increases. Accordingly, we hypothesized that the performance of our hybrid methods would lie between Expected Sarsa and GQ: being equal to Expected Sarsa when facing an on-policy problem, and growing ever closer to GQ as the problems become more off-policy.

In reality, we see that this hypothesis is close, but not always true. Looking at figure 7.1, we see that TDGQ's performance is worse than GQ on the inverted and dependent features problems in the extreme cases of the target policy $T10$. Remembering that TDGQ involves learning the set of weights learned by GQ to use in its own learning update, the poor performance of TDGQ may be caused by GQ's own difficulty with these problems. Again looking at figure 7.1, Hybrid-GQ and TDGQ perform worse than GQ in the dependent feature problems when the target policy is right dominated—with the probability of selecting the right action being 0.8 or higher. This may be attributable however to the effect of the value function initialization. As discussed in chapter 5, our initialization of θ means the agent begins with a highly optimistic value function, particularly in the inverted and dependent feature problems, and this in turn seems to make problems where the target policy heavily favours right actions easier.

In spite of this, Hybrid-GQ shows great promise in these results. Being a hybrid gradient-TD method, Hybrid-GQ automatically matches the performance of Expected Sarsa on on-policy problems, without any external intervention from the experimentor. Additionally, in these results we see that Hybrid-GQ practically dominates GQ in terms of performance on this problem. When we began this thesis, our two main goals were to find gradient-TD algorithms which could match conventional-TD level performance on on-policy problems, and to provide faster off-policy performance than existing gradient-TD methods. In these experiments, Hybrid-GQ has been shown to do both amply well, promising that not only is Hybrid-GQ an excellent gradient-TD method, but more generally, a fast RL method. While further

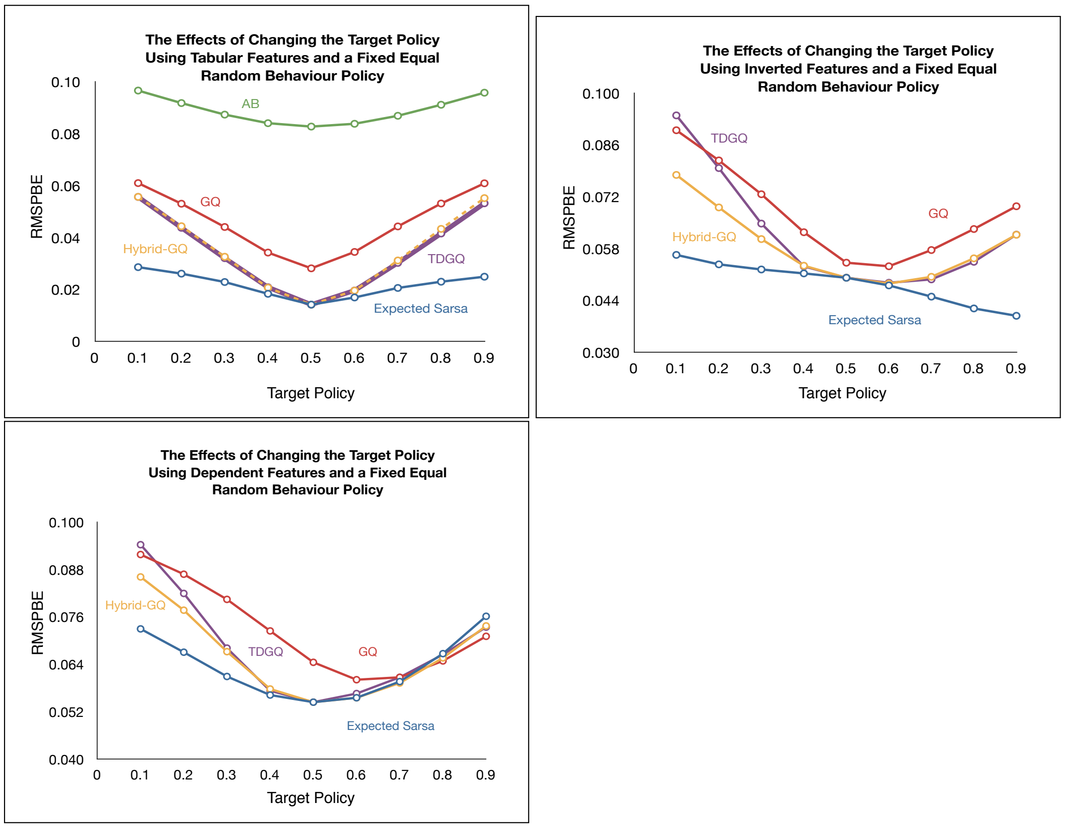

Figure 7.1: Using a fixed behaviour policy that chooses the actions left and right with equal probability, these graphs show how modifying the target policy (indicated as a probability of selecting the right action) affects the RMSPBE averaged over all 50 runs and 500 time-steps. The values for α and η used by each algorithm are the values which have the minimum mean RMSPBE averaged over all target polices.Standard error bars are left off the graph as the errors are smaller than the symbols used to mark data points. Results for AB are omitted for the inverted and dependent feature graphs as the algorithm performs significantly worse, dwarfing the differences between the other algorithms and making it difficult to interpret the graphs.

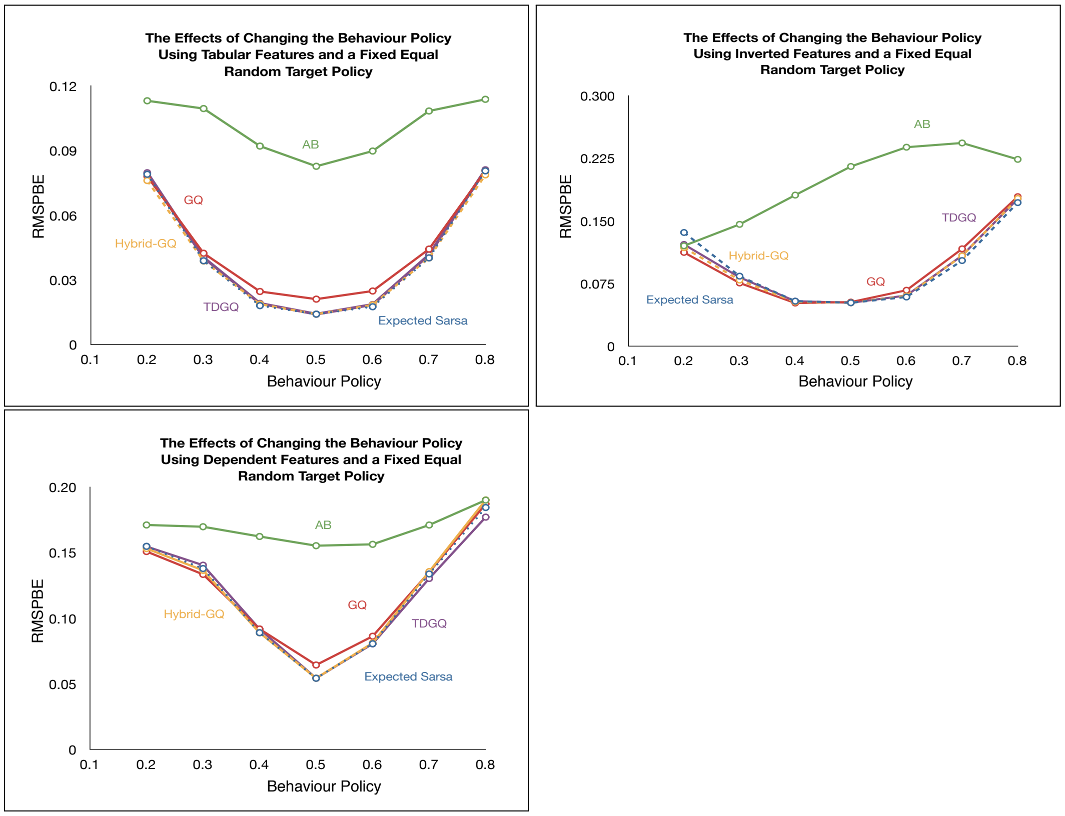

Figure 7.2: Using a fixed target policy that chooses the actions left and right with equal probability, these graphs show how modifying the behaviour policy affects the RMSPBE averaged over all 50 runs and 500 time-steps of our Hybrid Algorithms. The values for α and η are the values which have the minimum mean RMSPBE averaged over all behaviour polices. Standard error bars are left off the graph as the errors are smaller than the symbols used to mark data points.

results are necessary to make stronger claims, Hybrid-GQ shows great promise and calls for further experimentation.

Chapter 8

Conclusion

This thesis is about testing the limits of the new gradient-TD family of algorithms. While we recognize that gradient-TD algorithms have many desirable properties—providing a versatile, usable solution for the long-standing difficult problem of finding convergent a off-policy algorithm which works with function approximation—we question whether these methods can stand up against conventional-TD methods in terms of convergence speed when solving on-policy problems. We began this thesis with the belief that there is a need for, and a potential to develop, faster gradient-TD algorithms. Therefore and so we set out to find answers as to whether gradient-TD algorithms were truly slower than conventional-TD methods, and if so, by how much? Finally, could we find a way to overcome this slowness.

Our methodology focused on comparing GQ-NEU, GQ2, and GQ to one another and to compare each to Expected Sarsa on a simple Random Walk with actions environment, looking at on- and off-policy settings to determine how gradient-TD methods compared to basic conventional-TD methods. This is the first experimental work done with GQ-NEU and GQ2. We concurrently began introducing several variants of these algorithms: observing the effects of recomputing the TD-error between the updates for the two weight vectors of our gradient-TD methods in GQ-NEU+, GQ2+, and GQ+, and refactoring the GQ gradient to get a more sample-able gradient in Factored GQ and Double-Factored GQ. We found that recomputing the TD-error offered improvement on all our gradient-TD methods, with GQ+ being particularly effective. While results with GQ+ have been positive, there is a need for future work testing its effectiveness on a larger scale, with more difficult problems.

The second contribution of this thesis is the introduction of Hybrid Gradient-TD algorithms. Chapter 6 introduced the concept of algorithms with blended updates, which can automatically do a conventional-TD update on on-policy samples, but change its update to be increasingly more like gradient-TD methods when encountering off-policy data. We presented two new hybrid algorithms—Hybrid-GQ and

TDGQ—as well as a third related non-hybrid method AB. AB and Hybrid-GQ were hitherto previously described briefly in the appendix of Maei's thesis (2011), though we provide the first formal full description of the methods and their derivation here. This thesis is the first publication introducing TDGQ. We then then showed preliminary results for these three algorithms on the same off-policy chain-like MDP used throughout this thesis. These were the first results published for all three algorithms. From this experiment, we saw that Hybrid-GQ, being a hybrid gradient-TD method, was able to match Expected Sarsa's performance on on-policy problems. Also, Hybrid-GQ dominated GQ's performance on our off-policy problem, providing exciting results indicating that Hybrid-GQ is the fastest of the gradient-TD methods evaluated in this thesis. These highly favourable initial experiments show Hybrid-GQ to be a very promising hybrid gradient-TD algorithm, and a suitable answer to our call for faster gradient-TD methods.

References

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

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

Bellman, R. E. (1957). *Dynamic Programming.* Princeton University Press.

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

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

Maei, H. R. (2011). *Gradient Temporal-Difference Learning Algorithms.* PhD thesis, University of Alberta.

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

Rummery, G., and Niranjan, M. (1994). "On-line Q-learning using connectionist systems," Cambridge University, Tech. Rep. CUED/F-INFENC/TR 166, 1994.

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

Sutton, R. S. (1988). Learning to predict by the method of temporal differences.

Machine Learning, 3:9-44.

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

Sutton, R. S., Szepesvári, Cs., Maei, H. R. (2009). A convergent $\mathcal{O}(n)$ algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in Neural Information Processing Systems 21.* MIT Press.

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

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

van Seijen, H., van Hasselt, H. P., Whiteson, S., Wiering, M.A., (2009). A theoretical and empirical analysis of Expected Sarsa. *Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 177-184.

Watkins, C.J.C.H. (1989). Learning from delayed rewards. *Ph.D. Dissertation.* University of Cambridge.