Off-Policy Control under Changing Behaviour

Lucas Lehnert

School of Computer Science
McGill University, Montreal

August, 2016

Contents

Abstract

Reinforcement learning, as a part of machine learning, is the study of how to compute intelligent behaviour strategies that maximize a particular objective. Typically one considers an intelligent agent that interacts with its world and receives feedback in the form of a scalar value, called the reward. The goal is to find a behaviour strategy, called a policy, which maximizes an accumulative function of all rewards received during the agent's lifetime. Off-policy learning refers to the problem of learning a particular policy while using a different one to interact with the world. Gradient-based off-policy algorithms such as TDC [Sutton et al., 2009a] and GQ(λ) [Maei and Sutton, 2010] are incremental algorithms that are proven to converge even when used with linear function approximation. They have been developed for predicting the performance of a given fixed policy. However, in control problems one needs to search for a good policy by starting with a random policy and improving it over time. For the algorithm to incrementally improve a particular policy it needs to be able to adapt the policy during learning.

This thesis presents the first off-policy gradient-based learning algorithm that adapts the behaviour policy and accounts for how this effects the data the algorithm will see in the future. The algorithm, similar to the GQ(0) algorithm, is derived while leveraging ideas from policy gradient methods. Further, empirical evidence showing improved performance over existing approaches is presented.

Résumé

L'apprentissage par renforcement, un sous-domaine de l'apprentissage automatique, a pour but de parvenir à des comportements intelligents visant à maximiser un certain objectif. Typiquement, on considère l'existence d'un agent intelligent qui interagit avec le monde et reçoit de l'information sous forme d'un nombre appelé récompense. Le but est de trouver une stratégie comportementale, appelé politique, visant a maximiser les récompenses reçues durant la durée de vie de l'agent. "Off-policy learning" considère le probleme d'apprendre une politique en utilisant une autre pour interagir avec le monde. Les algorithmes "Gradient-based off-policy" (hors ligne de conduite et basé sur une dénivellation) comme TDC [Sutton et al., 2009a] et GQ(λ) [Maei and Sutton, 2010] sont des algorithmes itératifs qui ont une preuve de convergence même si utilisés avec des fonctions d'approximation linéaires. Ils ont été developpé pour prédire la performance d'une politique donnée. Par contre, dans les problèmes de contrôle, il faut chercher une bonne politique en commençant par une politique aléatoire puis en l'améliorant avec le temps. Pour que l'algorithme puisse améliorer itérativement une politique, il doit pouvoir adapter la politique au fur et à mesure de l'apprentissage.

Cette thèse présente le premier algorithme d'apprentissage "off policy, gradient based" qui ajuste la ligne de conduite comportementale et tient compte comment ces effets de données de l'algorithme seront perçus dans l'avenir. L'algorithme, similaire à l'algorithme GQ(0), s'inspire des idées des méthodes "policy gradient". De plus, des preuves empiriques montrant la meilleure performance de l'algorithme comparée aux performances des méthodes existantes sont présentées.

Acknowledgements

First, many thanks go to my advisor Doina Precup for her advice, encouragement, and guidance on my first step of how to be a researcher.

I would also like to thank Rich Sutton, Michael Littman, and Csaba Scepesvari for insightful discussions, showing me new aspects of my work, and for improving my research. My thanks also go to Pierre-Luc Bacon for many very helpful discussions. Further, special thanks go to Alan Do-Omri for his help with French translations and Valerie Burgardt for her linguistic advise and proofreading of this thesis.

Finally, I am grateful to my family for their constant loving support.

1

Introduction

Reinforcement learning (RL) is the study of how an intelligent agent interacts with its world to optimize some objective. The interaction between the agent and its world consists of the agent selecting an *action* which changes the *state* of the world. After such a state transition, the world provides the agent with a *reward*, a single scalar number. Ultimately, the goal of the agent is then to chose actions intelligently given the current state so that the total experienced reward is maximized.

A particular strategy specifying which action the agent chooses at which state is called a *policy*. Sutton and Barto [1998] provide different algorithms for approaching how to learn a policy. One concept used in many methods is that of a *value function* which estimates the expected return a particular policy can generate starting at a particular state in the system. Learning the value function of a particular policy is called the *prediction case*, whereas searching for a policy maximizing the total experienced reward is called the *control case*. The following sections summarize various approaches to learn a policy.

1.1 Finding Policies with a Known Model

Dynamic programming methods [Sutton and Barto, 1998, Chapter 4] are algorithms which compute an optimal policy or value function given a perfect model of the world. These methods do not learn through interacting with their world and instead have access to a perfect model. Computing the value function of a particular policy is called *policy evaluation*. In the subsequent *policy improvement* step, the policy is advanced using the model and

the value function computed during the previous policy evaluation step. Repeating policy evaluation followed by a policy improvement step until no further improvement can be made is called *policy iteration*.

Perkins and Precup [2003] introduced a convergent form of approximate policy iteration. This algorithm estimates the value function using transition data and subsequently improves the current policy using an improvement operator. Hence, a perfect model of the world is not required. These two steps are repeated until the improvement operator is not able to improve the policy any further. One key innovation of this work lies in the design of the improvement operator which generates a sequence of policies where changes in the policies are smooth. This smoothness property of the policy sequence allows to prove convergence of the method when a linear function is used to approximate the value function. However, this method is a policy iteration algorithm which first estimates the value function using a batch of transition samples and then improves the current policy. While it may be possible to approximate the value function with only one transition sample and improve the policy after every transition (which means the algorithm performs incremental online control), this algorithm does not evaluate and improve the current policy estimate simultaneously.

1.2 Learning from Temporal Differences

Temporal difference learning (TD-learning) is a class of algorithms that learns from an error signal called the *Temporal difference error* (TD-error). Sutton [1988] introduced the first TD-learning algorithm which estimates the value function of a fixed policy by finding the value function estimate that satisfies the *Bellman fixed point* [Bertsekas, 1996]. This algorithm calculates TD-errors to characterize how far the current value function estimate deviates from satisfying the Bellman fixed point. The major advantage of this approach is that TD-errors can be calculated from a single interaction between the agent and the world. Hence TD-learning algorithms are incremental and estimates can be updated after each interaction.

In TD-learning one distinguishes between two different use cases. One is called the *prediction case* where the algorithm estimates a value function for a specified fixed policy.

The other is called the *control case* where the algorithm simultaneously searches for an optimal policy and estimates its value function. A commonly used prediction algorithm is TD(0) [Sutton, 1988] which estimates the value function of a fixed policy as a function of only the state. The SARSA algorithm [Rummery, 1995] is a an extension of the TD(0) algorithm to the control case. SARSA also uses TD-errors to estimate the value function, but views it as a function of state-action pairs. Estimating the expected return for each state-action pair has the advantage that a policy can be directly computed form these estimates. Typically such a policy selects high valued actions with higher probability than low valued actions. Using the current value estimates, SARSA computes a policy to interact with its world. As the estimates vary over time, the policy used to select actions also varies over time. Singh et al. [2000] showed that under certain conditions this algorithm can find an optimal policy satisfying the *Bellman Optimality condition* [Bertsekas, 1996, Sutton and Barto, 1998]. However, SARSA may converge only to a region in which it may oscillate indefinitely [Gordon, 2001, 1996]. This phenomenon is called *chattering*.

TD-learning was also extended to the *off-policy learning* case. In Off-policy learning one learns a target policy while interacting with the agent's world according to a different behaviour policy. Being able to learn off-policy is very useful because the behaviour policy can be defined by a human user or stem from noisy data. Q-learning [Watkins and Dayan, 1992, Watkins, 1989] is an example of an off-policy control algorithm. This algorithm is similar to SARSA, however, Q-learning estimates a state-action value function for a target policy that deterministically selects the action of highest value. The policy used to interact with the agent's world may be any different policy. Jaakkola et al. [1994] proved that Q-learning converges to an optimal solution satisfying the Bellman Optimality condition. However, if function approximation is used to approximate the state-action value function, convergence cannot be guaranteed. In fact, on a classical domain known as the Baird counter example [Baird, 1995] Q-learning diverges monotonically [Maei et al., 2010].

Precup et al. [2001] presented the first provably convergent off-policy TD-learning algorithm. This algorithm uses products over importance sampling ratios to correct for the probability of sampling a trajectory under a behaviour policy that is different from the target policy. However, due to the use of products of importance sampling ratios, this method

suffers from high variance, thus making the algorithm impractical.

Gradient based TD-learning algorithms such as GTD(0) [Sutton et al., 2009b] and TDC [Sutton et al., 2009a] are a different class of algorithms that are stable under off-policy training whenever the value function is estimated using linear function approximation. The approach is different in that an objective is defined over the whole state and action space of the agent's world. Sutton et al. used this objective to derive a stochastic gradient descend algorithm similar to TD-learning. By the design of the objective function, which takes into account approximation errors and the probability of reaching a particular state, the obtained algorithm converges under off-policy learning and linear function approximation. Since the probability of reaching a particular state is explicitly modelled in the objective function, products over importance sampling ratios are not necessary to correct the probability of seeing a particular trajectory so far, but they are still used for correcting the probability of a single transition. Thus, gradient based TD-learning algorithms are more stable in comparison to the algorithm presented by Precup et al. [2001]. The extension of these methods to the control case are the GQ(λ) [Maei and Sutton, 2010] and GreedyGQ [Maei et al., 2010] algorithms. However, these algorithms are only guaranteed to converge if the behaviour policy is fixed. As we will show in Chapter 4, on two standard control problems GQ(0) is not able to converge to an efficient control policy if its behaviour policy is allowed to vary between time steps. The requirement of a fixed behaviour policy makes these algorithm unsuitable for control problems where a good policy is unknown and needs to be discovered by the control algorithm.

1.3 Directly Maximizing Return

Another approach to learning a good policy is the use of policy gradients which was first introduced by Sutton et al. [2000]. Such a method assumes the policy to be a differentiable function of a particular parameter vector. The objective one considers is the expected total return a particular parameter vector can generate. Sutton et al. present a policy iteration algorithm which maximizes the total return using policy gradients. While this method is provably convergent, it requires to find an approximation of the value function before the

policy parameters can be improved with a gradient update. Depending on the implementation and application scenario, it may be computationally expensive to find an approximation of the value function or one may have to sample a batch of transition data to compute an approximation. Nevertheless, the policy can be non-linear in its parameters and so powerful function approximation methods, especially deep neural networks, can be employed.

Actor Critic methods are another approach to learning a good policy [Sutton and Barto, 1998, Chapter 6.6]. They make an explicit distinction between the policy used to select actions, called the *actor*, and the value function used to evaluate the policy, called the *critic*. These methods also incorporate TD-errors to update the critic which then updates the actor to improve the current policy.

Degris et al. [2012] introduce an off-policy Actor-Critic algorithm which uses GTD(λ) to learn the value function of the critic. Since gradient based TD-learning methods are used, they prove convergence in the control case when linear function approximation is used. However, Thomas [2014] shows that the gradient update is biased and that for certain cases the convergence proof does not hold.

1.4 Contributions

In this thesis we introduce a new gradient based control algorithm similar to the GQ algorithm which incorporates policy gradients, by approaching the algorithm derivation similar to policy gradient methods. The key innovation is to consider the policy to be an operator on top of the value function, which allows the algorithm to search more directly in the space of all possible control policies by accounting for the interaction between the control policy and the distribution with which interaction data is sampled. Existing methods such as Q-learning, SARSA, and GQ do not consider how changing the control policy changes the probability of making a particular transition. Our method is the first algorithm that considers this interaction between control policy and the probability of making a particular transition, which results in performance and stability improvements over existing methods. The idea of considering the policy to be merely an operator on top of the value function is similar to the idea of using an improvement operator in the approximate policy iteration algorithm

mentioned in Section 1.3. Ultimately our analysis yields the first algorithm that performs policy evaluation and improvement simultaneously and can adapt its policy incrementally after every interaction between the agent and the world.

First the technical foundations are presented in Chapter 2, followed by the derivation of the algorithm in Chapter 3. Chapter 4 shows empirically that the presented method remains stable when used with linear function approximation for off-policy control, similar to existing gradient based TD-learning algorithms. Further, we show that our method also outperforms existing gradient based TD-learning algorithms on a set of standard benchmark domains by finding a more efficient policy. Chapter 5 concludes with a discussion of how the presented research can be continued.

2

Reinforcement Learning

This chapter presents some fundamental concepts in sequential decision making and RL. Based on these concepts, different approaches to learn efficient policies in the context of RL are introduced.

2.1 Markov Decision Processes

A discrete-time stochastic process is a sequence of random variables (r.v.s) X_t indexed by a time parameter $t \in \mathbb{N} = \{0, 1, 2, ...\}$. The value of the random variable is called the *state* of the process and the space of all possible states is denoted with $\mathcal{S}$. One key purpose of stochastic processes is to model phenomena that are correlated between consecutive time steps. Rather than viewing the sequence or r.v.s as a family of identically and independently distributed (i.i.d.) random variables one often uses them to model correlations between different time steps. If the distribution of a random variable X_t is dependent only on the outcome of the random variable at the previous time step X_{t-1}, then we say that the discrete-time stochastic process fulfills the *Markov property*. Such a stochastic process is called a *Markov chain*.

Definition 1 (Markov Chain)**.** Let $\mathcal{S}$ be a set of possible outcomes and $\{X_t | t \in \mathbb{N}\}$ be a discrete-time stochastic process where X_t takes values $x_t \in \mathcal{S}$. The stochastic process $\{X_t | t \in \mathbb{N}\}$ is called a Markov chain if it fulfills the Markov property

$$\forall t \geq 0,\ \mathbf{P}\{X_{t+1} = x_{t+1} | X_0 = x_0, ..., X_t = x_t\} = \mathbf{P}\{X_{t+1} = x_{t+1} | X_t = x_t\}.$$

The random variable X_0 is independently distributed to any of the other random variables.

Intuitively, this definition means that a Markov chain is a discrete-time stochastic process where the first random variable X_0 follows a distribution over a state space $\mathcal{S}$ and the distribution of any X_{t+1} solely depends on the outcome of X_t at the previous time step t. The distribution of X_0 is often referred to as the *start state distribution*.

Another requirement can be imposed on a Markov chain such that the probabilities $\mathbf{P}\{X_{t+1} = x_{t+1}|X_t = x_t\}$ do not change over time. Thus the distribution of the next state X_{t+1} conditioned on X_t does not change with time index t. Such a Markov chain is called *stationary*.

Definition 2 (Stationary Markov Chain)**.** A Markov Chain $\{X_t|t \in \mathbb{N}\}$ with a state space $\mathcal{S}$ is called stationary if

$$\forall t \geq 0, \forall r \geq 0,\ \mathbf{P}\{X_{t+1} = y|X_t = x\} = \mathbf{P}\{X_{r+1} = y|X_r = x\}, \tag{2.1}$$

for some $x, y \in \mathcal{S}$.

A Markov chain whose transition probabilities $\mathbf{P}\{X_{t+1} = x_{t+1}|X_t = x_t\}$ change with the time index t is called *non-stationary*. Suppose the state space is a finite set $\mathcal{S} = \{s_1, ..., s_n\}$ and the transition probabilities are denoted with $p_{i,j} = \boldsymbol{P}\{X_{t+1} = s_j|X_t = s_i\}$. Then a transition matrix for a stationary Markov chain is

$$\boldsymbol{P} = \begin{bmatrix} p_{1,1} & \cdots & p_{n,1} \\ \vdots & \ddots & \vdots \\ p_{1,n} & \cdots & p_{n,n} \end{bmatrix}. \tag{2.2}$$

This matrix is also called a *Markov matrix*. It is useful to use this notation when we think about random walks through a Markov chain. For example, consider n different states where the process starts at state 1. The start state distribution is modelled as an n dimensional vector $\boldsymbol{d}_0 = [1, 0, ...0]^\top$ whose entries sum to one. This means one reasons about a probability distribution over the state space $\mathcal{S}$ with all mass concentrated at state 1. Suppose one transition is made in the Markov chain. Using the transition matrix the distribution over states after one time step can be computed with

$$\boldsymbol{d}_1 = \boldsymbol{P}\boldsymbol{d}_0.$$

After t time steps the distribution over the state space is

$$\boldsymbol{d}_t = \underbrace{(\boldsymbol{P} \cdots \boldsymbol{P})}_{t \text{ times}} \boldsymbol{d}_0.$$

This transition matrix $\boldsymbol{P}$ can also be viewed as an operator on a probability distribution over the finite state space $\mathcal{S}$. The fixed point of such a Markov matrix is called the *stationary distribution.*

Definition 3 (Stationary Distribution)**.** Let $\{X_t | t \in \mathbb{N}\}$ be a Markov chain with a finite state space $\mathcal{S} = \{s_1, ..., s_n\}$ and transition matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n}$. The vector $\boldsymbol{d} = [p_1, ..., p_n]^\top$ whose entries lie in the interval $[0, 1]$ and sum to one is called the stationary distribution of $\{X_t | t \in \mathbb{N}\}$ if

$$\boldsymbol{d} = \boldsymbol{P}\boldsymbol{d}. \tag{2.3}$$

Thus far a mathematical model has been introduced that represents how a system can change between its states according to some stochastic transition dynamics. To be able to represent an intelligent agent in such a framework, an agent needs to be able to make decisions. The ability of making decisions is implemented in this system by adding a set of actions $\mathcal{A}$ which is assumed to be finite in this thesis. In this model, making a decision corresponds to selecting an action from the set $\mathcal{A}$. Since the agent should be able to control its state through selecting actions, the transition dynamics of the Markov chain are dependent on the selected actions through conditioning the transition probabilities on the selected action as well as the state of the system. The probability of reaching a specific state is denoted with $\mathbf{P}\{X_{t+1} = s_{t+1} | X_t = s_t, A_t = a_t\}$ where the selected action is modelled as the random variable A_t. The agent chooses actions according to a policy which is a distribution over the space of actions conditioned on the state. Typically a policy is denoted with π and the probability of selecting an action is denoted with

$$\pi(a|s) = \mathbf{P}\{A_t = a | S_t = s\}. \tag{2.4}$$

If the agent selects actions deterministically, then the probability of selecting one action is set to one whereas all other action selection probabilities are zero.

To this point a framework was introduced where an agent chooses actions to control the transition model of a Markov chain on a step by step basis. One additional concept is

needed: the objective the agent can optimize. This is implemented through a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ which gives a scalar reward to the agent after every transition. The objective of the agent is then to find a policy π that maximizes the experienced reward. This framework is called a *Markov Decision Process*.

Definition 4 (Markov Decision Process (MDP))**.** A finite action discounted Markov Decision Process (MDP) is a quintuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ where the set $\mathcal{S}$ is the state space, the finite set $\mathcal{A}$ is the action space, the transition function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is defined as

$$p(s_t, a_t, s_{t+1}) = \mathbf{P}\{X_{t+1} = s_{t+1} | X_t = s_t, A_t = a_t\}, \tag{2.5}$$

and the function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function. The discount factor $\gamma \in [0, 1)$ and if $\gamma = 1.0$ the MDP is un-discounted.

The discount factor γ has the effect of favouring short term rewards over long term rewards by weighting the reward at time step t with a factor γ^{t-1}. If one looks at infinite-length trajectories using a discount factor of $\gamma < 1$ is necessary to ensure that the total return, the sum of all received rewards weighted by γ^{t-1}, remains bounded for any bounded reward function. Section 2.1.1 discusses this in more detail.

Suppose an agent operates within an MDP and follows a fixed policy π. In this case, the probability of transitioning from a state s_t to state s_{t+1} is

$$\mathbf{P}\{X_{t+1} = s_{t+1} | X_t = s_t\} = \sum_{a \in \mathcal{A}} \pi(a|s_t) p(s_t, a, s_{t+1}). \tag{2.6}$$

The fixed policy π together with the transition dynamics of the MDP defines a Markov chain over the state space $\mathcal{S}$. Since the agent still collects rewards while transitioning between states, a *Markov Reward Process* emerges.

Definition 5 (Markov Reward Process (MRP))**.** A finite action discounted Markov Reward Process (MRP) is a quadruple $\mathcal{R} = \langle \mathcal{S}, p, r, \gamma \rangle$ where the set $\mathcal{S}$ is the state space, the transition function $p : \mathcal{S} \times \mathcal{S} \to [0, 1]$ is defined as

$$p(s_t, s_{t+1}) = \mathbf{P}\{X_{t+1} = s_{t+1} | X_t = s_t\}, \tag{2.7}$$

and the function $r : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ is the reward function. The discount factor $\gamma \in [0, 1)$ and if $\gamma = 1.0$ then the MRP is un-discounted.

2.1.1 Value Functions: Quantifying the Quality of a Behaviour

The goal of RL is to find algorithms that can efficiently compute a good policy π directly from interactions between the agent and its environment. This interaction data consists of trajectories through the MDP which can have finite or infinite length. A finite trajectory is a sequence of transition tuples

$$\tau = \{(s_t, a_t, s_{t+1})\}_{t=0}^{T}.$$

The state s_0 is referred to as the start state and actions are sampled according to the action selection probabilities of the policy π. The next state is then sampled according to the transition function p of the MDP. Given a transition (s_t, a_t, s_{t+1}), the reward $r_{t+1} = r(s_t, a_t, s_{t+1})$ is deterministic. Often the reward function is considered to be a function of state-action pairs defined as

$$r_E(s, a) = \mathop{\mathbb{E}}_{s' \sim p(s,a,\cdot)} [r(s, a, s')] = \sum_{s' \in \mathcal{S}} p(s, a, s') r(s, a, s'). \tag{2.8}$$

According to Sutton et al. [2000], the quality of a policy can be characterized by the total cumulative return

$$\rho(\pi) = \lim_{T \to \infty} \frac{1}{T} \mathbb{E} \left[r_0 + r_1 + \cdots + r_T | s_0 \in \mathcal{S}, \pi \right], \tag{2.9}$$

where the expectation is over all possibe trajectories τ of length T. The discounted cumulative return is defined as

$$\rho_\gamma(\pi) = \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0 \in \mathcal{S}, \pi \right], \tag{2.10}$$

where the expectation is over all possible trajectories τ of infinite length starting at state s_0. The discounted cumulative return can be viewed as a function of the start state giving rise to the *value function* [Sutton and Barto, 1998] defined as

$$V^\pi(s) = \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0 = s, \pi \right]. \tag{2.11}$$

The superscript denotes the dependence of the value function on the policy. By applying a weight γ^{t-1} to every reward one can favour short-term rewards over long-term rewards by picking a γ close to zero. Since the summation is a series taking t from one to infinity, this weighting scheme also ensures that values remain bounded for a bounded reward function without having to make further assumptions on the transition dynamics of the MDP.

In the context of planning within an MDP, a value function characterizes the usefulness or utility of a specific state to the agent. The generalization of this idea to state-action pairs gives rise to the *action-value function* (also referred to as *state-action value function* or *Q-function*) and is defined as

$$\begin{aligned} Q^\pi(s,a) &= \mathop{\mathbb{E}}_{s'\sim p(s,a,\cdot)}\left[r(s,a,s') + \gamma V^\pi(s')\right] \\ &= r_E(s,a) + \gamma \mathop{\mathbb{E}}_{s'\sim p(s,a,\cdot)}\left[V^\pi(s')\right]. \end{aligned} \tag{2.12}$$

An optimal policy for starting at state s in a given MDP $\mathcal{M}$ is defined as

$$\pi^* = \arg\sup_\pi V^\pi(s). \tag{2.13}$$

The value functions under an optimal policy are denoted as

$$V^*(s) = \sup_\pi V^\pi(s), \quad Q^*(s,a) = \sup_\pi Q^\pi(s,a). \tag{2.14}$$

The following two theorems [Sutton and Barto, 1998] further characterize an optimal policy based on its value function.

Theorem 1 (Bellman Equations)**.** Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ be an MDP with policy π, then

$$\forall s \in \mathcal{S}, \qquad V^\pi(s) = \mathop{\mathbb{E}}_{\substack{a\sim\pi(\cdot|s) \\ s'\sim p(s,a,\cdot)}}\left[r(s,a,s') + \gamma V^\pi(s')\right], \tag{2.15}$$

$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A}, \qquad Q^\pi(s,a) = \mathop{\mathbb{E}}_{s'\sim p(s,a,\cdot)}\left[r(s,a,s') + \gamma V^\pi(s')\right]. \tag{2.16}$$

Theorem 2 (Bellman Optimality)**.** Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ be an MDP with policy π, then π is an optimal policy for $\mathcal{M}$ if and only if for all $s \in \mathcal{S}$, π selects an action

$$a^* \in \arg\max_{a\in\mathcal{A}} Q^*(s,a) \tag{2.17}$$

with probability one. An action $b \notin \arg\max_{a\in\mathcal{A}} Q^*(s,a)$ is selected with probability zero.

Using Theorem 1 the Bellman optimality condition can be written for action-value functions as

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s'\sim p(s,a,\cdot)}\left[r(s,a,s') + \gamma \max_b Q^*(s',b)\right]. \tag{2.18}$$

This condition is also called the *Bellman fixed point*.

The following sections review several algorithms to tackle two common problems in RL: the prediction problem where the value function of a fixed policy is estimated and the control problem where an optimal policy is computed.

2.1.2 Vector and Function Notation

Aside from the experiments presented in Chapter 4, the state and action spaces are considered to be both finite. Further, the ith entry of a vector $\boldsymbol{v}$ is denoted with $\boldsymbol{v}(i)$. If the vector $\boldsymbol{v}$ is indexed by states or state-action pairs, the entry corresponding to state s is denoted with $\boldsymbol{v}(s)$ for the former case and the entry corresponding to s, a is denoted with $\boldsymbol{v}(s, a)$ for the latter case. For a matrix $\boldsymbol{A}$ the entry at the ith row and jth column is denoted $\boldsymbol{A}(i, j)$, the ith row is denoted with $\boldsymbol{A}(i, \cdot)$, and the jth row is denoted with $\boldsymbol{A}(\cdot, j)$. Vector norms are denoted with $||\cdot||$. The L2 norm of a vector $\boldsymbol{v}$ is defined as

$$||\boldsymbol{v}|| = ||\boldsymbol{v}||_2 = \sqrt{\sum_i |\boldsymbol{v}(i)|^2}.$$

The L1 norm of a vector $\boldsymbol{v}$ is defined as

$$||\boldsymbol{v}||_1 = \sum_i |\boldsymbol{v}(i)|.$$

Lastly, the infinity norm of a vector is defined as

$$||\boldsymbol{v}||_\infty = \max_i |\boldsymbol{v}(i)|.$$

Previously, the value functions have been defined as a function mapping the state space or state-action space to the reals. Since we assume their domain to be finite one can also represent these functions as a vector indexed by the state or state-action space. In the following chapters this tabular definition and the functional definition are used interchangeably as they are equivalent.

2.2 Temporal Difference Learning

One central goal in RL is to develop algorithms that can learn a good policy π through interacting with their environment. In this online learning scenario, the agent has to update

its internal estimates every time it selects an action and observes a transition and reward. One distinguishes between the prediction case, where one estimates the value function for a fixed policy, and the control case, where one simultaneously evaluates the current policy and improves it. In addition, in the control case the policy with which actions are selected varies between consecutive time steps. Thus, the distribution from which transitions are sampled changes from time step to time step.

Suppose we have an agent interacting with its environment according to a fixed policy π. By Theorem 1

$$V^{\pi}(s) = \mathbb{E}\left[R(s, a, s') + \gamma V^{\pi}(s')\right].$$

Suppose there is a probability distribution over the state space $\mathcal{S}$ and that the agent arrives at a state $s \in \mathcal{S}$ with probability p_s. Then one can write

$$\begin{aligned} & \sum_{s \in \mathcal{S}} p_s V^{\pi}(s) = \sum_{s \in \mathcal{S}} p_s \mathbb{E}\left[r(s, a, s') + \gamma V^{\pi}(s')\right] \\ \implies & \sum_{s \in \mathcal{S}} p_s \mathbb{E}\left[r(s, a, s') + \gamma V^{\pi}(s') - V^{\pi}(s)\right] = 0 \\ \implies & \mathbb{E}\left[r(s, a, s') + \gamma V^{\pi}(s') - V^{\pi}(s)\right] = 0, \end{aligned} \tag{2.19}$$

where the expectation in the last line ranges over triplets (s, a, s') and actions are selected according to the fixed policy π. Using (2.19) we can define the *Temporal Difference error* (TD-error) for any transition (s, a, s') as

$$\delta = r(s, a, s') + \gamma V^{\pi}(s') - V^{\pi}(s). \tag{2.20}$$

The TD(0) algorithm [Sutton, 1988] is an incremental algorithm that estimates the value function V^{π} for a particular fixed policy π by sampling transition triplets (s, a, s') and then moving the current value function estimate towards a fixed point that gives a zero TD-error. Using moving average updates a particular value function estimate $V_{t+1}(s)$ is computed with

$$\begin{aligned} V_{t+1}(s) &= (1 - \alpha) V_t(s) + \alpha \left[r_t + \gamma V_t(s')\right] \\ &= V_t(s) + \alpha \left[r_t + \gamma V_t(s') - V_t(s)\right] \\ &= V_t(s) + \alpha \delta, \end{aligned} \tag{2.21}$$

for some learning rate $\alpha \in (0, 1]$. These updates are applied until the estimates have converged and $\delta = 0$, resulting in a value function estimate satisfying the Bellman fixed point condition (2.15) from Theorem 1. Algorithm 1 gives a listing of TD(0).

Algorithm 1 TD(0) Sutton [1988], Sutton and Barto [1998]

Input: An MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, a policy π, a learning rate $\alpha \in (0, 1]$, a start state s_{start}.
  Initialize $\forall s \in \mathcal{S}$, $V^{\pi}(s) = 0$.
  $s \leftarrow s_{\text{start}}$
  repeat
    Sample $a \sim \pi(\cdot|s)$
    Take action a, observe reward r and next state s'
    $V(s) \leftarrow V(s) + \alpha \left[r + \gamma V(s') - V(s) \right]$
    $s \leftarrow s'$
  until state s is terminal

TD(0) is considered a prediction algorithm because actions are selected with a fixed policy π. Consider now the control case in which a policy π needs to be found that maximizes the total return the agent receives. In this case, the same concepts are generalized to state-action value functions and we define the TD-error based on a quintuple (s, a, r, s', a') as

$$\delta = r(s, a, s') + \gamma Q(s', a') - Q(s, a), \tag{2.22}$$

which gives rise to the update rule of SARSA [Rummery, 1995, Sutton and Barto, 1998]

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r(s, a, s') + \gamma Q(s', a') - Q(s, a) \right]. \tag{2.23}$$

Note that if the policy π is fixed, then SARSA makes the same updates as TD(0) does but for a function defined on $\mathcal{S} \times \mathcal{A}$. The advantage of updating an action-value function is that the action values can be directly used to calculate a control policy. Typically, either an ε-greedy policy (Definition 6) or a Boltzmann policy (Definition 7) is used.

Definition 6 (ε-Greedy Policy)**.** Given an action-value function Q, a policy π is called ε-Greedy if it selects an action a at state s with probability

$$\pi(a|s) = \begin{cases} (1 - \varepsilon)\frac{1}{m} & \text{if } a \in \arg\max_a Q(s, a) \\ \varepsilon \frac{1}{|\mathcal{A}|} & \text{otherwise,} \end{cases} \tag{2.24}$$

where $m = |\{a | a \in \arg\max_a Q(s, a)\}|$ (the number of actions of highest value) and $\varepsilon \in [0, 1]$ is a parameter.

Definition 7 (Boltzmann/Softmax Policy)**.** Given an action-value function Q, a policy π is called Boltzmann or Softmax if it selects an action a at state s with probability

$$\pi(a|s) = \frac{\exp\left(Q(s,a)/\tau\right)}{\sum_{b\in\mathcal{A}} \exp(Q(s,b)/\tau)}, \tag{2.25}$$

where $\tau > 0$ is a temperature parameter.

SARSA is a control algorithm which uses the current action-values to compute its control policy and updates them using (2.23). Algorithm 2 gives a listing of SARSA. Expected SARSA [Sutton and Barto, 1998, Exercise 6.10] is a variation of SARSA which computes the expectation of the action selected at the next state s' analytically rather than sampling the action. The update rule of Expected SARSA is then

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r(s,a,s') + \gamma \mathbb{E}_{a'}[Q(s',a')] - Q(s,a)\right], \tag{2.26}$$

where $\mathbb{E}_{a'}[Q(s',a')] = \sum_{a'} \pi(a'|s')Q(s',a')$. By computing the expectation analytically Expected SARSA is often more stable than SARSA in practice.

Algorithm 2 SARSA Rummery [1995], Sutton and Barto [1998]

Input: An MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, a learning rate $\alpha \in (0, 1]$, a start state s_{start}.
  Initialize $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}$, $Q(s,a) = 0$.
  $s \leftarrow s_{\text{start}}$
  Choose a at s using a policy derived from Q
  repeat
    Take action a, observe reward r and next state s'
    Choose a' at s' using a policy derived from Q
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$, $a \leftarrow a'$
  until state s is terminal

Q-learning [Watkins, 1989], another popular control algorithm, updates its Q-function according to the rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{b\in\mathcal{A}} Q(s',b) - Q(s,a)\right]. \tag{2.27}$$

Q-learning differs from SARSA in that the update rule uses a max-operator. Since the next action of highest Q-value is used to make the update (irrespective of what the current policy π is), Q-learning is considered an off-policy learning algorithm as it learns the Q-function for

a greedy policy while actions are selected according to a possibly different policy π. We will discuss off-policy learning in more detail in Section 2.3. The use of the max-operator can be motivated by the Bellman optimality condition (2.16) as one tries to match the update term to the target estimate more directly. Algorithm 3 shows a full listing of Q-learning.

Algorithm 3 Q-learning Watkins [1989], Sutton and Barto [1998]

Input: An MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, a learning rate $\alpha \in (0, 1]$, a start state s_{start}.
  Initialize $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}$, $Q(s, a) = 0$.
  $s \leftarrow s_{\text{start}}$
  Choose a at s using a policy derived from Q
  repeat
    Take action a, observe reward r and next state s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{b \in \mathcal{A}} Q(s', b) - Q(s, a) \right]$
    $s \leftarrow s'$
    Choose a at s using a policy derived from Q
  until state s is terminal

Both Q-learning and SARSA are proven to converge to an optimal policy if the value function is represented as a table [Jaakkola et al., 1994, Singh et al., 2000]. For SARSA to converge to an optimal policy satisfying the Bellman optimality condition (2.16), the control policy must become greedy with respect to the action-values in the limit. Singh et al. [2000] call this class of policies *Greedy in the limit with infinite exploration (GLIE).*

Definition 8 (GLIE - Greedy in the limit with infinite exploration)**.** A policy is GLIE if it satisfies the following conditions:

1. Every state-action pair is visited infinitely often.
2. In the limit, the policy selects the action of highest Q-value with probability one (w.p.1).

Singh et al. [2000] provide conditions so that the ε-greedy and Boltzmann policies fall into the category of GLIE policies. Intuitively for one such policy to become more greedy one needs to decay either the ε parameter or temperature τ over time. More specifically, if a Boltzmann policy is used, Singh et al. prove that if the temperature parameter at time step t and state s is $\tau_t(s) \propto 1/\ln n_t(s)$ ($n_t(s)$ is the number of times state s was visited at step t) then the policy is GLIE. Similarly if an ε-greedy policy is used, one requires $\varepsilon_t(s) = c/n_t(s)$ for $c \in (0, 1)$ for the policy to satisfy the GLIE conditions. Under these conditions the SARSA algorithm converges to an optimal policy. The intuition behind using GLIE policies

is that SARSA makes updates that are more and more similar to Q-learning as time passes and the policy becomes more and more greedy. In the limit, both Q-learning and SARSA become equivalent.

2.2.1 Function Approximation

In order to scale these methods to application domains with large state spaces, a common approach is to use function approximation to approximate the value function. We will consider only the case of linear function approximation where the value functions are approximated with

$$V_{\boldsymbol{\theta}}(s) = \boldsymbol{\theta}^\top \phi(s) \approx V^\pi(s), \quad Q_{\boldsymbol{\theta}}(s,a) = \boldsymbol{\theta}^\top \psi(s,a) \approx Q^\pi(s,a), \tag{2.28}$$

where $\phi : \mathcal{S} \to \mathbb{R}^n$ and $\psi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^n$ are basis functions. The form of these basis functions as well as the dimension n of the parameter vector $\boldsymbol{\theta}$, is typically adapted to each application domain separately.

The update rule (2.21) of the TD(0) algorithm can be rewritten to use linear function approximation [Sutton and Barto, 1998] with

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha\delta\nabla_{\boldsymbol{\theta}} V_{\boldsymbol{\theta}}(s) \\ &= \boldsymbol{\theta}_t + \alpha\delta\phi(s), \end{aligned} \tag{2.29}$$

where the TD-error is computed as

$$\delta = r(s,a,s') + \gamma\boldsymbol{\theta}^\top\phi(s') - \boldsymbol{\theta}^\top\phi(s).$$

Similarly, one can also rewrite the update rule (2.23) of the SARSA algorithm as

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha[r(s,a,s') + \gamma Q_{\boldsymbol{\theta}}(s',a') - Q_{\boldsymbol{\theta}}(s,a)]\nabla_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}(s,a) \\ &= \boldsymbol{\theta}_t + \alpha[r(s,a,s') + \gamma\boldsymbol{\theta}^\top\psi(s',a') - \boldsymbol{\theta}^\top\psi(s,a)]\psi(s,a). \end{aligned} \tag{2.30}$$

Sutton and Barto [1998, Chapter 8] provide a motivation explaining this way of using the gradient of the value function.

One can also define the basis functions as zero-one bit vectors $\phi(s) \in \mathbb{R}^{|\mathcal{S}|}$ and $\psi(s,a) \in \mathbb{R}^{|\mathcal{S}\times\mathcal{A}|}$ having zero entries everywhere except for the entry corresponding to the state s or

state-action pair s, a being set to one. It is easy to verify that this case corresponds to the tabular versions (Algorithm 2 and 1) defined previously. We refer to the use of such basis functions as the *tabular case*.

2.3 Off-Policy Learning

Off-policy learning refers to the problem of learning a policy different from the policy that is used to generate the transition data. The ability to reuse data from one policy to learn a different policy has many advantages. For example, this is useful in the context of robotics in order to learn a good policy from noisy data. In control, off-policy learning is also used to facilitate exploration. The Q-learning algorithm presented in the previous section (Algorithm 3) is an example of an off-policy learning algorithm. Here the policy used to generate the transition samples is arbitrary, whereas the algorithm updates its Q-function for a policy that is greedy with respect to the Q-values because the max operator is used for computing the next state-action Q-value. Typically the policy used to generate the transition data makes exploratory random moves to ensure that the whole state-action space is sampled and that the data from which the greedy target policy is learned is representative. This technique facilitates exploration and allows the algorithm to converge to an optimal policy faster.

The policy used for generating transition data is referred to as the *control policy* or *behavior policy* b and the policy the agent tries to learn is referred to as the *target policy* π. For example, for Q-learning one could choose an ε-greedy policy as the control policy b in order to learn a greedy target policy π. Note that the greedy target policy is hard coded into the Q-learning algorithm.

2.3.1 Importance Sampling

Suppose a trajectory $\tau = \{(s_t, a_t, s_{t+1})\}_{t=0}^{T}$ is generated on an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ by some behaviour policy b and let $p(s_0)$ denote the probability of starting τ at state s_0. Under

b, the probability of this trajectory to occur is then

$$\mathbf{P}\{\tau|b\} = p(s_0)\prod_{t=0}^{T} b(a_t|s_t)p(s_t, a_t, s_{t+1}). \tag{2.31}$$

Under a different target policy π, this trajectory is sampled with probability

$$\begin{aligned}\mathbf{P}\{\tau|\pi\} &= p(s_0)\prod_{t=0}^{T} \pi(a_t|s_t)p(s_t, a_t, s_{t+1}) \\ &= p(s_0)\prod_{t=0}^{T} b(a_t|s_t)\frac{\pi(a_t|s_t)}{b(a_t|s_t)}p(s_t, a_t, s_{t+1}) \\ &= \mathbf{P}\{\tau|b\}\prod_{t=0}^{T}\frac{\pi(a_t|s_t)}{b(a_t|s_t)} \\ &= \mathbf{P}\{\tau|b\}\prod_{t=0}^{T}\rho_t,\end{aligned} \tag{2.32}$$

where the *importance sampling ratio*

$$\rho_t \overset{\text{def.}}{=} \frac{\pi(a_t|s_t)}{b(a_t|s_t)}. \tag{2.33}$$

Sometimes ρ_t is also called the *importance sampling correction*.

2.3.2 Off-policy Temporal Difference Learning

Using importance sampling ratios the TD(0) algorithm can be rewritten as an off-policy learning algorithm with the update rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\rho\delta\phi(s), \tag{2.34}$$

where $\rho = \pi(a|s)/b(a|s)$. However, on the Baird counter example [Baird, 1995] TD(0) as well as Q-learning do not remain stable since their weights diverge monotonically to infinity [Sutton et al., 2009a, Maei and Sutton, 2010].

Precup et al. [2001] derive a version of the TD(0) algorithm that accumulates the importance sampling ratios over time to correct for the probability of the whole trajectory. This version of TD(0) is provably convergent under off-policy learning with linear function approximation. However, the algorithm suffers from high variance in the value function estimation due to the use of products over importance sampling ratios.

In the next section gradient TD methods will be presented. These methods no longer correct the probability of reaching a particular state with products over importance sampling ratios. As a result, gradient TD methods do not suffer from high variance in the value function estimation. Moreover, they are stable under off-policy training with linear function approximation.

2.4 Gradient Based TD Methods

Gradient-based TD-learning algorithms such as GTD [Sutton et al., 2009b], GTD2, and TDC [Sutton et al., 2009a] are the first incremental off-policy TD-learning algorithms that remain stable if used with linear-function approximation. As previously stated, the value function is approximated with

$$V_{\boldsymbol{\theta}}(s) = \boldsymbol{\theta}^\top \phi(s) \approx V^\pi(s), \quad Q_{\boldsymbol{\theta}}(s,a) = \boldsymbol{\theta}^\top \psi(s,a) \approx Q^\pi(s,a).$$

Let $||\boldsymbol{v}||^2_{\boldsymbol{D}} = \boldsymbol{v}^\top \boldsymbol{D}\boldsymbol{v}$ be a weighted L2 norm where $\boldsymbol{D} = \text{diag}\{d(s)\}_{s\in\mathcal{S}}$ is a $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix. GTD2 and TDC both perform stochastic gradient descent on the *Mean Squared Projected Bellman Error (MSPBE)* [Sutton et al., 2009a] objective defined as

$$\text{MSBPE}(\boldsymbol{\theta}) = ||\boldsymbol{V}_{\boldsymbol{\theta}} - \Pi T^\pi \boldsymbol{V}_{\boldsymbol{\theta}}||^2_{\boldsymbol{D}}, \tag{2.35}$$

where $\boldsymbol{V}_{\boldsymbol{\theta}} \in \mathbb{R}^{|\mathcal{S}|}$ is a vector listing all state-values $V_{\boldsymbol{\theta}}(s)$, and T^π is the Bellman operator defined as

$$T^\pi \boldsymbol{v} \stackrel{\text{def.}}{=} \boldsymbol{R}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v}, \tag{2.36}$$

where $\boldsymbol{R}^\pi \in \mathbb{R}^{|\mathcal{S}|}$ is a vector whose entries contain the expected reward $R^\pi(s) = \mathbb{E}[r(s,a,s')|s]$ (the expectation is over the random variables $a \sim \pi(\cdot|s)$ and $s' \sim p(s,a,\cdot)$) and $\boldsymbol{P}^\pi$ is a state-to-state transition matrix under a policy π. The projection operator is defined as the matrix

$$\Pi = \boldsymbol{\Phi}\left(\boldsymbol{\Phi}^\top \boldsymbol{D}\boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^\top \boldsymbol{D},$$

where $\boldsymbol{\Phi}$ is a matrix with each row containing the basis function vector $\phi(s)$ for every state s. This projection operator projects the one-step lookahead $T^\pi \boldsymbol{V}_{\boldsymbol{\theta}}$ back into the space of

all representable value functions. The use of this projection operator makes linear function approximation stable because it performs a linear regression fit of the linear model to the value function defined on the finite state space $\mathcal{S}$ (every state s can be thought of as one data point in the domain of the value function that is fitted). Finding the parameter vector $\boldsymbol{\theta}$ which minimizes the MSPBE($\boldsymbol{\theta}$) corresponds to finding the best approximation to the true value function V^π.

One key innovation of the MSPBE objective is the use of the weight matrix $\boldsymbol{D}$ which applies a weight $d(s)$ to the expected projected Bellman error at every state s. These weights are interpreted as the probability with which a state is visited and are called the *stationary distribution*. This distribution is invariant of the time step t in the simulated trajectory and the start state of the trajectory. This use of the stationary distribution in conjunction with the projection operator Π stabilizes TDC and GTD2 under off-policy training when it is used with linear function approximation.

2.4.1 Derivation of TDC

To derive an incremental algorithm minimizing the MSPBE using transition samples of the form $(s, r(s,a,s), s')$, the MSPBE objective is rewritten as

$$\text{MSBPE}(\boldsymbol{\theta}) = ||V_{\boldsymbol{\theta}} - \Pi T^\pi V_{\boldsymbol{\theta}}||^2_{\boldsymbol{D}} = \mathbb{E}[\delta\phi^\top]\mathbb{E}[\phi\phi^\top]^{-1}\mathbb{E}[\delta\phi], \tag{2.37}$$

where we define $\boldsymbol{\phi} := \phi(s)$ and $\boldsymbol{\phi}' := \phi(s')$. The gradient of the MSPBE is then

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}}\text{MSBPE}(\boldsymbol{\theta}) &= 2\nabla_{\boldsymbol{\theta}}\mathbb{E}[\delta\boldsymbol{\phi}^\top]\mathbb{E}[\boldsymbol{\phi}\boldsymbol{\phi}^\top]^{-1}\mathbb{E}[\delta\boldsymbol{\phi}] \\
&= 2\nabla_{\boldsymbol{\theta}}\mathbb{E}[(r(s,a,s') + \gamma\boldsymbol{\theta}^\top\boldsymbol{\phi}' - \boldsymbol{\theta}^\top\boldsymbol{\phi})\boldsymbol{\phi}^\top]\mathbb{E}[\boldsymbol{\phi}\boldsymbol{\phi}^\top]^{-1}\mathbb{E}[\delta\boldsymbol{\phi}] \\
&= 2\mathbb{E}[(\gamma\boldsymbol{\phi}' - \boldsymbol{\phi})\boldsymbol{\phi}^\top]\mathbb{E}[\boldsymbol{\phi}\boldsymbol{\phi}^\top]^{-1}\mathbb{E}[\delta\boldsymbol{\phi}] \\
&= 2(\mathbb{E}[\gamma\boldsymbol{\phi}'\boldsymbol{\phi}^\top] - \mathbb{E}[\boldsymbol{\phi}\boldsymbol{\phi}^\top])\mathbb{E}[\boldsymbol{\phi}\boldsymbol{\phi}^\top]^{-1}\mathbb{E}[\delta\boldsymbol{\phi}] \\
&= 2\mathbb{E}[\gamma\boldsymbol{\phi}'\boldsymbol{\phi}^\top]\mathbb{E}[\boldsymbol{\phi}\boldsymbol{\phi}^\top]^{-1}\mathbb{E}[\delta\boldsymbol{\phi}] - 2\mathbb{E}[\delta\boldsymbol{\phi}] \\
\iff -\frac{1}{2}\nabla_{\boldsymbol{\theta}}\text{MSBPE}(\boldsymbol{\theta}) &= \mathbb{E}[\delta\boldsymbol{\phi}] - \mathbb{E}[\gamma\boldsymbol{\phi}'\boldsymbol{\phi}^\top]\mathbb{E}[\boldsymbol{\phi}\boldsymbol{\phi}^\top]^{-1}\mathbb{E}[\delta\boldsymbol{\phi}]
\end{aligned} \tag{2.38}$$

Since the expectations of the gradient have to be sampled independently, the weight doubling trick first presented in Sutton et al. [2009b] is used and an auxiliary weight vector is defined:

$$\boldsymbol{w} \overset{\text{def.}}{=} \mathbb{E}[\boldsymbol{\phi}\boldsymbol{\phi}^\top]^{-1}\mathbb{E}[\delta\boldsymbol{\phi}]. \tag{2.39}$$

This auxiliary weight vector $\boldsymbol{w}$ can then be estimated with the update rule

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \beta_t(\delta - \boldsymbol{\phi}^\top \boldsymbol{w}_t)\boldsymbol{\phi}. \tag{2.40}$$

This update rule allows the gradient of the MSPBE to be sampled directly to update the parameter vector $\boldsymbol{\theta}$ with

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t(\delta\boldsymbol{\phi} - \gamma\boldsymbol{\phi}'\boldsymbol{\phi}^\top \boldsymbol{w}_t). \tag{2.41}$$

For this two-time scale stochastic approximation algorithm [Borkar, 1997, Sutton et al., 2009a] the learning rates are typically set so that $\alpha_t \ll \beta_t$. Intuitively, this condition on the learning rates means that the sequence $\{\boldsymbol{w}_t\}$ converges to its solution $\boldsymbol{w}$ faster than the sequence $\{\boldsymbol{\theta}_t\}$ does. As a result, $\boldsymbol{\theta}_t$ appears as almost steady when $\boldsymbol{w}$ is estimated and similarly $\boldsymbol{w}_t$ appears as the correct auxiliary weight (as it converges quickly) when $\boldsymbol{\theta}_t$ is updated. To convert these update rules to the off-policy case, we consider the batch gradient to select actions according to the behaviour policy b. The expectations of the gradient, which are with respect to the behaviour policy b (this is denoted in the subscript of the expectation operator), are then corrected with an importance sampling ratio ρ:

$$-\frac{1}{2}\nabla_{\boldsymbol{\theta}}\text{MSPBE}(\boldsymbol{\theta}) = \mathbb{E}_b[\rho\delta\boldsymbol{\phi}] - \mathbb{E}_b[\rho\gamma\boldsymbol{\phi}'\boldsymbol{\phi}^\top]\mathbb{E}_b[\rho\boldsymbol{\phi}\boldsymbol{\phi}^\top]^{-1}\mathbb{E}_b[\rho\delta\boldsymbol{\phi}]. \tag{2.42}$$

To sample this gradient, the update rules (2.41), (2.40) need to include the importance sampling ratio ρ. Algorithm 4 shows a full listing of the resulting algorithm which is called TDC.

This algorithm is proven to converge to the correct value function parameter $\boldsymbol{\theta}$ under off-policy training with linear function approximation if the samples $(s, r(s, a, s'), s')$ are i.i.d. For the convergence proof, Sutton et al. [2009a] use the ODE method [Borkar and Meyn, 1999] and reduce the algorithm to work on a Markov Reward Process. This proof method works well in the prediction case as the policy used to generate transitions is fixed.

2.4.2 Gradient Based Q-learning

Maei and Sutton [2010] generalize the TDC algorithm to estimate an action-value function and derive a gradient based Q-learning algorithm called GQ(λ). The λ parameter in the name is used for eligibility traces [Sutton and Barto, 1998, Chapter 7], which are an extension

Algorithm 4 TDC Sutton et al. [2009a]

Input: An MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, a target policy π, a behaviour policy b, a learning rate $\alpha \in (0, 1]$, a start state s_{start}.
Initialize $\boldsymbol{\theta} = 0$, $\boldsymbol{w} = 0$.
$s \leftarrow s_{\text{start}}$, $t \leftarrow 1$
repeat
Sample $b \sim \pi(\cdot|s)$
Take action a, observe reward r and next state s'
$\rho \leftarrow \frac{\pi(a|s)}{b(a|s)}$
$\boldsymbol{\phi} \leftarrow \phi(s)$, $\boldsymbol{\phi}' \leftarrow \phi(s')$
$\delta \leftarrow r + \gamma \boldsymbol{\theta}^\top \boldsymbol{\phi}' - \boldsymbol{\theta}^\top \boldsymbol{\phi}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_t \rho (\delta \boldsymbol{\phi} - \gamma \boldsymbol{\phi}' \boldsymbol{\phi}^\top \boldsymbol{w})$
$\boldsymbol{w} \leftarrow \boldsymbol{w} + \beta_t \rho (\delta - \boldsymbol{\phi}^\top \boldsymbol{w}) \boldsymbol{\phi}$
$s \leftarrow s'$, $t \leftarrow t + 1$
until state s is terminal

of the algorithm to make more efficient updates and which is disabled when $\lambda = 0$. Since we do not consider eligibility traces in this thesis, we will consider an algorithm without the eligibility trace extension which is equivalent to GQ(0). The derivation of this algorithm is almost identical to the derivation of TDC. GQ(0) is also referred to as just GQ. For gradient based Q-learning, one considers the stationary distribution to be a distribution over the state-action space $\mathcal{S} \times \mathcal{A}$ (now the diagonal matrix $\boldsymbol{D}$ is of size $|\mathcal{S} \times \mathcal{A}| \times |\mathcal{S} \times \mathcal{A}|$) and defines the MSPBE objective as

$$\text{MSPBE}(\boldsymbol{\theta}) \stackrel{\text{def.}}{=} ||\boldsymbol{Q}_{\boldsymbol{\theta}} - \Pi T^\pi \boldsymbol{Q}_{\boldsymbol{\theta}}||^2_{\boldsymbol{D}}. \tag{2.43}$$

The Bellman operator T^π and the projection operator

$$\Pi = \boldsymbol{\Psi}(\boldsymbol{\Psi}^\top \boldsymbol{D} \boldsymbol{\Psi})^\top \boldsymbol{\Psi}^\top \boldsymbol{D}$$

are still defined in the same way, but now a matrix $\boldsymbol{\Psi}$ containing the basis vectors $\psi(s, a)$ as rows for all state-action pairs is used. Moreover, the transition matrix $\boldsymbol{P}^\pi$ is now of size $|\mathcal{S} \times \mathcal{A}| \times |\mathcal{S} \times \mathcal{A}|$ and maps state-action pairs to state-action pairs (rather than states to states). The reward vector $\boldsymbol{R}^\pi \in \mathbb{R}^{|\mathcal{S} \times \mathcal{A}|}$ has entries $R^\pi(s, a) = \mathbb{E}[r(s, a, s')|s, a]$ (the expectation is only over the random variable $s' \sim p(s, a, \cdot)$). For the prediction case, GQ has the same convergence guarantees and stability properties that TDC has. However, GQ is only proven to converge if the behaviour and target policies are fixed. Further, we will show in Chapter 4 that on some standard control domains GQ(0) is not able to converge to an

efficient control policy if its control policy is allowed to vary between time steps, making GQ(0) unsuitable for control problems where a good control policy is unknown and has to be discovered by the algorithm. Algorithm 5 gives a listing of GQ, which is the same as the TDC algorithm but uses a basis function ψ defined on the state-action space.

Algorithm 5 GQ(0), adopted from Maei and Sutton [2010]

Input: An MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, a target policy π, a behaviour policy b, a learning rate $\alpha \in (0, 1]$, a start state s_{start}.
Initialize $\boldsymbol{\theta} = 0$, $\boldsymbol{w} = 0$.
$s \leftarrow s_{\text{start}}$, $t \leftarrow 1$
repeat
  Sample $b \sim \pi(\cdot|s)$
  Take action a, observe reward r and next state s'
  $\rho \leftarrow \frac{\pi(a|s)}{b(a|s)}$
  $\boldsymbol{\psi} \leftarrow \psi(s, a)$, $\overline{\boldsymbol{\psi}} \leftarrow \sum_b \pi(b|s')\psi(s', b)$
  $\delta \leftarrow r + \gamma \boldsymbol{\theta}^\top \overline{\boldsymbol{\psi}} - \boldsymbol{\theta}^\top \boldsymbol{\psi}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_t \rho (\delta \boldsymbol{\psi} - \gamma \overline{\boldsymbol{\psi}} \boldsymbol{\psi}^\top \boldsymbol{w})$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \beta_t \rho (\delta - \boldsymbol{\psi}^\top \boldsymbol{w}) \boldsymbol{\psi}$
  $s \leftarrow s'$, $t \leftarrow t + 1$
until state s is terminal

Maei et al. [2010] present GreedyGQ, a version of the GQ algorithm where the target policy π is greedy with respect to the action-value function estimate. They prove convergence of the GreedyGQ algorithm with a non-stationary target policy π; however, they still assume a fixed behavior policy b. The proof analyses the GreedyGQ algorithm as a Two Timescale Stochastic Approximation algorithm [Borkar, 1997]. Since the ODE method [Borkar and Meyn, 1999] is used, the policy b is required to be fixed as the transition samples (s, a, s') are assumed to be i.i.d. However, their proof is more evolved than the convergence proof of GQ because the target policy is assumed to be parametric in the value function parameter $\boldsymbol{\theta}$, which makes it non-stationary.

2.5 Policy Iteration Methods

This section explains how dynamic programming (DP) algorithms perform policy evaluation and policy search. Consider an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ with a finite state space $\mathcal{S}$ and

a finite action space $\mathcal{A}$. In this case, the value function $V^\pi(s)$ is presented as a vector $\boldsymbol{V}^\pi$ indexed by states. Similarly, a reward vector $\boldsymbol{R}^\pi \in \mathbb{R}^{|\mathcal{S}|}$ can be defined where each entry corresponds to

$$\mathbb{E}_{a,s'}[r(s,a,s')|s] = \sum_{a,s'} \pi(a|s)p(s,a,s')r(s,a,s')$$

and a transition matrix $\boldsymbol{T}^\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ where each entry is defined by

$$p_{i,j} = \mathbf{P}\{s_j|s_i\} = \sum_{a\in\mathcal{A}} \pi(a|s_i)p(s_i,a,s_j),$$

where $s_i, s_j \in \mathcal{S}$ and i and j are integers indexing the finite state space. Using these matrix definitions we can rewrite the Bellman Equation (2.15) as

$$\boldsymbol{V}^\pi = \boldsymbol{R}^\pi + \gamma \boldsymbol{T}^\pi \boldsymbol{V}^\pi. \tag{2.44}$$

Since there is a closed form expression for the value function, it can be directly computed simply by rearranging the previous equation:

$$\boldsymbol{V}^\pi = (\boldsymbol{I} - \gamma \boldsymbol{T}^\pi)^{-1} \boldsymbol{R}^\pi. \tag{2.45}$$

Ng and Russell [2000] show that the matrix $\boldsymbol{I} - \gamma \boldsymbol{T}^\pi$ is always invertible and therefore the closed form expression can be used to compute the value function directly. However this method is not particularly efficient as we have to compute the full transition matrix and reward vector and then invert a matrix which can be very large if the state space of the MDP is large.

2.5.1 Iterative Policy Evaluation

An alternative to this method is an algorithm called Iterative Policy Evaluation [Sutton and Barto, 1998]. This algorithm initializes the value function vector with the zero vector and then repeatedly iterates over the state space to push the value function estimates to the Bellman fixed point (2.15). Algorithm 6 shows a listing of Iterative Policy Evaluation.

2.5.2 Policy Iteration

Policy iteration improves a policy iteratively until an optimal policy is obtained. Suppose a policy deterministically chooses actions, i.e. it is a function mapping $\pi : \mathcal{S} \to \mathcal{A}$, and its

Algorithm 6 Iterative Policy Evaluation adopted from Sutton and Barto [1998]

Input: A policy π, an MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, a small number $\varepsilon > 0$.
  $\forall s \in \mathcal{S}$, $V^\pi(s) \leftarrow 0$.
  repeat
    $\Delta \leftarrow 0$
    for all $s \in \mathcal{S}$ **do**
      $v \leftarrow V^\pi(s)$
      $V^\pi(s) \leftarrow \sum_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \pi(a|s) p(s, a, s')[r(s, a, s') + \gamma V^\pi(s')]$
      $\Delta \leftarrow \max\{\Delta, |V^\pi(s) - v|\}$
    end for
  until $\Delta < \varepsilon$

corresponding action-value function $Q^\pi(s, a)$.[1] The current policy π can be improved by always forcing it to greedily select the action of highest value, i.e. the improved policy π_{new} is such that

$$\forall s \in \mathcal{S},\ \pi_{\text{new}}(s) = \arg\max_{a \in \mathcal{A}} Q^\pi(s, a). \tag{2.46}$$

Then Iterative Policy Evaluation is run again on the new policy π_{new} to obtain the new action-value function $Q^{\pi_{\text{new}}}$. Sutton and Barto [1998] prove that this repeated alternation between policy improvement and policy evaluation will always converge to an optimal policy. Algorithm 7 shows a listing of Policy Iteration.

Algorithm 7 Policy Iteration adopted from Sutton and Barto [1998]

Input: MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$
  Randomly initialize π
  policyStable $\leftarrow$ false
  while not policyStable **do**
    $\boldsymbol{V}^\pi \leftarrow$ IterativePolicyEvaluation$(\pi, \mathcal{M})$
    for all $s \in \mathcal{S}$ **do**
      $\pi_{\text{new}}(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s'} p(s, a, s')[r(s, a, s') + \gamma V^\pi(s')]$
    end for
    policyStable $\leftarrow$ is $\pi = \pi_{\text{new}}$?
    $\pi \leftarrow \pi_{\text{new}}$
  end while
  return π

[1] This can also be obtained by setting one action selection probability to one and all others to zero for each state.

2.5.3 Approximate Policy Iteration

Perkins and Precup [2003] present the first policy iteration algorithm that learns from interaction data (now a perfect model of the world is not necessary) which is provably convergent when used with linear function approximation. Being a policy iteration algorithm, this method performs first policy evaluation using the SARSA algorithm to learn the value function and then improves the current policy using policy improvement. The key idea of this work is to use a policy improvement operator Γ which maps action values to a policy which is ε-soft. For an action value vector $\boldsymbol{Q}$ the policy $\Gamma(\boldsymbol{Q})$ chooses an action at random with probability ε. If ε is decreased slowly towards zero with every improvement step, the algorithm will converge to an optimal policy satisfying the Bellman fixed point. This decrease in ε is similar to using a ε-greedy GLIE policy, except that the parameter is decayed after every policy improvement step (rather than with every transition the SARSA algorithm makes, as described previously). Algorithm 8 shows a listing of Approximate Policy Iteration.

Algorithm 8 Approximate Policy Iteration from Perkins and Precup [2003]

Input: MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, an initial policy π, a basis function ψ with matrix $\boldsymbol{\Psi}$ containing $\psi(s,a)$ at every row, a policy improvement operator Γ, a start state s_{start}

```
s ← s_start
repeat
    Initialize θ arbitrarily.
    Chose action a ~ π(·|s)
    repeat
        Take action a, observe reward r and next state s'
        Chose action a' ~ π(·|s')
        θ ← θ + α [r + γ θ^⊤ ψ(s', a') − θ^⊤ ψ(s, a)] ψ(s, a)
        s ← s', a ← a'
    until θ does not change
    π ← Γ(Ψθ)
until π does not change
return π
```

While this algorithm is provably convergent, it distinguishes between an evaluation and improvement step. Further this is an on-policy learning algorithm. Many more policy iteration methods have been developed, see Bertsekas [2011] for an overview. All these methods sequentially alternate between policy evaluation and policy improvement.

2.6 Policy Gradient Methods

Policy gradient methods were first introduced by Sutton et al. [2000] who approached the problem of finding a good policy by directly maximizing the total expected return. Sutton et al. [2000] introduced an algorithm for the un-discounted and discounted case; however, this section only presents the discounted case. Consider a discounted MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$. If a policy $\pi_{\boldsymbol{\theta}}$ is used, the discounted total return starting at state s_0 is defined by

$$\rho_\gamma(\pi_{\boldsymbol{\theta}}) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0, \pi_{\boldsymbol{\theta}}\right],$$

where the policy $\pi_{\boldsymbol{\theta}}$ is assumed to be parametrized by the vector $\boldsymbol{\theta}$. In this formulation, a weighting over the state space is used which is defined as

$$d^{\pi_{\boldsymbol{\theta}}}(s) = \sum_{t=0}^{\infty} \gamma^t \mathbf{P}\{s_0 \to s, t, \pi_{\boldsymbol{\theta}}\}, \tag{2.47}$$

where $\mathbf{P}\{s_0 \to s, t, \pi_{\boldsymbol{\theta}}\}$ is the probability of transitioning from state s_0 to state s in t steps while following the policy $\pi_{\boldsymbol{\theta}}$. The policy gradient theorem then states the gradient of $\rho_\gamma(\pi)$ with respect to $\boldsymbol{\theta}$.

Theorem 3 (Policy Gradient adopted from Sutton et al. [2000])**.** For any MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, the gradient of $\rho_\gamma(\pi)$ is

$$\nabla_{\boldsymbol{\theta}} \rho_\gamma(\pi) = \sum_{s \in \mathcal{S}} d^{\pi_{\boldsymbol{\theta}}}(s) \sum_{a \in \mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a), \tag{2.48}$$

where $Q^{\pi_{\boldsymbol{\theta}}}$ is the state-action value function of $\pi_{\boldsymbol{\theta}}$.

Proof. First the gradient of the value function $V^{\pi_{\boldsymbol{\theta}}}$ with respect to $\boldsymbol{\theta}$ is considered for any state $s \in \mathcal{S}$:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} V^{\pi_{\boldsymbol{\theta}}}(s) &\overset{\text{def.}}{=} \nabla_{\boldsymbol{\theta}} \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a) \\
&= \sum_{a \in \mathcal{A}} \left[\nabla_{\boldsymbol{\theta}} \pi(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a) + \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} Q^{\pi_{\boldsymbol{\theta}}}(s, a) \right] \\
&= \sum_{a \in \mathcal{A}} \left[\nabla_{\boldsymbol{\theta}} \pi(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a) + \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \sum_{s' \in \mathcal{S}} \left[r(s, a, s') + \gamma p(s, a, s') V^{\pi_{\boldsymbol{\theta}}}(s') \right] \right] \\
&= \sum_{a \in \mathcal{A}} \left[\nabla_{\boldsymbol{\theta}} \pi(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a) + \pi_{\boldsymbol{\theta}}(a|s) \sum_{s' \in \mathcal{S}} \gamma p(s, a, s') \nabla_{\boldsymbol{\theta}} V^{\pi_{\boldsymbol{\theta}}}(s') \right]
\end{aligned} \tag{2.49}$$

The unrolling of (2.49) can be repeated an arbitrary number of times to obtain

$$\nabla_{\boldsymbol{\theta}} V^{\pi_{\boldsymbol{\theta}}}(s) = \sum_{s \in \mathcal{S}} \sum_{t=0}^{\infty} \gamma^t \mathbf{P}\{s_0 \to s, t, \pi_{\boldsymbol{\theta}}\} \sum_{a \in \mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a). \tag{2.50}$$

Using the definition of ρ_γ and $V^{\pi_{\boldsymbol{\theta}}}$ (see (2.11)) gives

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \rho_\gamma(\pi_{\boldsymbol{\theta}}) &= \nabla_{\boldsymbol{\theta}} \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \middle| s_0, \pi_{\boldsymbol{\theta}} \right] \\ &= \nabla_{\boldsymbol{\theta}} V^{\pi_{\boldsymbol{\theta}}}(s_0) \\ \implies \nabla_{\boldsymbol{\theta}} \rho_\gamma(\pi_{\boldsymbol{\theta}}) &= \sum_{s \in \mathcal{S}} \sum_{t=0}^{\infty} \gamma^t \mathbf{P}\{s_0 \to s, t, \pi_{\boldsymbol{\theta}}\} \sum_{a \in \mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a), \end{aligned}$$

as desired. □

Using this theorem, Sutton et al. [2000] derive a convergent policy iteration algorithm which uses function approximation for evaluating the current policy. Further, only a mild compatibility condition between the policy and the function approximation model is required. This requirement means that non-linear function approximation such as deep neural networks can be used with this algorithm. However, the derived method, while provably convergent, needs to compute an approximation of the value function before the policy can be improved with a gradient update. Since finding an approximation of the value function involves solving a system of equations, this step may be computationally expensive depending on the application scenario.

3

Off-Policy Control

The GQ algorithm is an extension of the TDC algorithm to the control case and is derived in the same way as its prediction counterpart. Hence GQ is only proven to converge if both target and control policy are fixed, making the algorithm unsuitable for control problems where a policy is unknown. In the control case the policy used to generate transition data is typically computed from the current value function estimate. If linear function approximation is used, the policies depend on the current estimate $\boldsymbol{\theta}_t$ which is updated from time step to time step. These variations in the value function parameter $\boldsymbol{\theta}_t$ cause the policies computed from them to change over time, resulting in a change in distribution with which transition data is generated. Specifically, in the control case, the sampled transition data is not i.i.d. solely due to variations in the value function parameter.

This observation can be used to derive a new gradient based TD-learning algorithm that takes into account the interaction between changes in the policy and the value function parameter. Hence, the Bellman operator T^π, as well as the stationary distribution over state-action pairs, both depend on $\boldsymbol{\theta}$:

$$\text{MSPBE}(\boldsymbol{\theta}) = ||Q_{\boldsymbol{\theta}} - \Pi_{\boldsymbol{\theta}} T_{\boldsymbol{\theta}} Q_{\boldsymbol{\theta}}||^2_{\boldsymbol{D}_{\boldsymbol{\theta}}}. \tag{3.1}$$

The Bellman operator parametric in $\boldsymbol{\theta}$ is defined as

$$T_{\boldsymbol{\theta}} \boldsymbol{v} \stackrel{\text{def.}}{=} \boldsymbol{R} + \gamma \boldsymbol{P}_{\boldsymbol{\theta}} \boldsymbol{v}. \tag{3.2}$$

The entry $\boldsymbol{P}_{\boldsymbol{\theta}}((s', a'), (s, a)) = p(s, a, s')\pi_{\boldsymbol{\theta}}(a'|s')$ is the probability of transitioning from (s, a) to (s', a') under the current policy $\pi_{\boldsymbol{\theta}}$. Since the policy depends on $\boldsymbol{\theta}$, the matrix $\boldsymbol{P}_{\boldsymbol{\theta}}$ and

the Bellman operator $T_{\boldsymbol{\theta}}$ depend on the parameter vector as well. By parametrizing the Bellman operator in this way the one step lookahead considers how changing the value function estimate changes the policy. As a result, one can update the parameter vector $\boldsymbol{\theta}$ to simultaneously improve and evaluate the current policy estimate. This approach does not consider policy evaluation and policy improvement as two separate steps. Hence, the algorithm we will derive in Section 3.1 performs policy evaluation and policy improvement simultaneously.

For the stationary distribution over state-action pairs, a stationary distribution $d(s)$ over the state space is assumed. Similar to GTD2 and TDC, $d(s)$ describes the probability of reaching a state s. Using this distribution we define

$$d_{\boldsymbol{\theta}}(s,a) \overset{\text{def.}}{=} d(s)\pi_{\boldsymbol{\theta}}(a|s),\ \boldsymbol{D}_{\boldsymbol{\theta}} = \text{diag}\{d_{\boldsymbol{\theta}}(s,a)\}_{(s,a)\in\mathcal{S}\times\mathcal{A}}. \tag{3.3}$$

Suppose the algorithm runs for t time steps and generates a parameter sequence $\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_t$ and with that a policy sequence $\pi_1, ..., \pi_t$. Under this sequence of policies, $d_t(s)$, the probability of visiting a particular state s after exactly t time steps, can be computed. Viewing $d(s)$ as steady is then equivalent to averaging $d_t(s)$ across all time steps. With this new parametrization of the MSPBE objective, its gradient is re-derived in the next section. Note that this parametrization of the current policy in terms of the parameter vector $\boldsymbol{\theta}$ is similar to the parametrization considered when deriving the Policy Gradient Theorem 3. However, we consider the MSPBE as an objective and assume the target policy $\pi_{\boldsymbol{\theta}}$ to be a differentiable operator of the value function $Q_{\boldsymbol{\theta}}$, similar to the policy improvement operator used for approximate policy iteration in Algorithm 8.

3.1 Derivation of the PGQ Algorithm

First, the MSPBE is written as the matrix equation

$$\text{MSPBE}(\boldsymbol{\theta}) = \left(\boldsymbol{\Psi}^\top D_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right)^\top (\boldsymbol{\Psi}^\top D_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1} \left(\boldsymbol{\Psi}^\top D_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right). \tag{3.4}$$

To compute gradients of an equation containing matrices as intermediate results, the partial derivatives with respect to $\boldsymbol{\theta}(i)$ are computed first:

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\theta}(i)}\text{MSPBE}(\boldsymbol{\theta}) &= \frac{\partial}{\partial \boldsymbol{\theta}(i)}\Big[\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right)^\top (\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}QT_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right)\Big] \\ &= 2\frac{\partial}{\partial \boldsymbol{\theta}(i)}\left[\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right)^\top\right](\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right) \\ &\quad + \left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right)^\top \frac{\partial}{\partial \boldsymbol{\theta}(i)}\left[(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\right]\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right).\end{aligned} \tag{3.5}$$

The derivative of the inverse feature covariance is

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\theta}(i)}(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1} &= -(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\frac{\partial}{\partial \boldsymbol{\theta}(i)}(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1} \\ &= -(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}(\boldsymbol{\Psi}^\top \frac{\partial \boldsymbol{D}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}\boldsymbol{\Psi})(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}.\end{aligned} \tag{3.6}$$

Plugging this back into the gradient gives

$$\begin{aligned}&\frac{\partial}{\partial \boldsymbol{\theta}(i)}\text{MSPBE}(\boldsymbol{\theta}) \\ &= 2\left(\boldsymbol{\Psi}^\top \frac{\partial \boldsymbol{D}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}}) + \boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\frac{\partial}{\partial \boldsymbol{\theta}(i)}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right)^\top (\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right) \\ &\quad - \left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right)^\top (\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}(\boldsymbol{\Psi}^\top \frac{\partial \boldsymbol{D}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}\boldsymbol{\Psi})(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right).\end{aligned} \tag{3.7}$$

The partial derivative on the Bellman error is

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\theta}(i)}[T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}}] &= \frac{\partial}{\partial \boldsymbol{\theta}(i)}[\boldsymbol{R} + \gamma \boldsymbol{P}_{\boldsymbol{\theta}}\boldsymbol{\Psi}\boldsymbol{\theta} - \boldsymbol{\Psi}\boldsymbol{\theta}] \\ &= \gamma \frac{\partial \boldsymbol{P}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}\boldsymbol{\Psi}\boldsymbol{\theta} + \gamma \boldsymbol{P}_{\boldsymbol{\theta}}\boldsymbol{\Psi}(\cdot, i) - \boldsymbol{\Psi}(\cdot, i).\end{aligned} \tag{3.8}$$

Plugging (3.8) back into the MSPBE gradient gives

$$
\begin{aligned}
&\frac{\partial}{\partial \boldsymbol{\theta}(i)}\text{MSPBE}(\boldsymbol{\theta}) \\
&= 2\Big(\boldsymbol{\Psi}^\top \frac{\partial \boldsymbol{D}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}}) \\
&\quad + \boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(\gamma \frac{\partial \boldsymbol{P}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}\boldsymbol{\Psi}\boldsymbol{\theta} + \gamma \boldsymbol{P}_{\boldsymbol{\theta}}\boldsymbol{\Psi}(\cdot, i) - \boldsymbol{\Psi}(\cdot, i))\Big)^\top (\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right) \\
&\quad - \left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right)^\top (\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}(\boldsymbol{\Psi}^\top \frac{\partial \boldsymbol{D}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}\boldsymbol{\Psi})(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right) \\
&= 2\left(\boldsymbol{\Psi}^\top \frac{\partial \boldsymbol{D}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right)^\top (\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right) \\
&\quad + 2\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(\gamma \frac{\partial \boldsymbol{P}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}\boldsymbol{\Psi}\boldsymbol{\theta} + \gamma \boldsymbol{P}_{\boldsymbol{\theta}}\boldsymbol{\Psi}(\cdot, i) - \boldsymbol{\Psi}(\cdot, i))\right)^\top (\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right) \\
&\quad - \left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right)^\top (\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}(\boldsymbol{\Psi}^\top \frac{\partial \boldsymbol{D}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}\boldsymbol{\Psi})(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right).
\end{aligned}
\tag{3.9}
$$

Similar to Sutton et al. [2009a] an auxiliary weight vector is defined as

$$
\boldsymbol{w} = (\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}\boldsymbol{\Psi})^{-1}\left(\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})\right) = \mathbb{E}[\boldsymbol{\Psi}\boldsymbol{\Psi}^\top]^{-1}\mathbb{E}[\delta\boldsymbol{\Psi}], \tag{3.10}
$$

to rewrite the partial derivative as

$$
\begin{aligned}
&\frac{1}{2}\frac{\partial}{\partial \boldsymbol{\theta}(i)}\text{MSPBE}(\boldsymbol{\theta}) \\
&= \boldsymbol{\Psi}\frac{\partial \boldsymbol{D}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}})^\top \boldsymbol{w} + \boldsymbol{\Psi}\boldsymbol{D}_{\boldsymbol{\theta}}\left(\gamma \frac{\partial \boldsymbol{P}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}\boldsymbol{\Psi}\boldsymbol{\theta} + \gamma \boldsymbol{P}_{\boldsymbol{\theta}}\boldsymbol{\Psi}(\cdot, i) - \boldsymbol{\Psi}(\cdot, i)\right)^\top \boldsymbol{w} - \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Psi}^\top \frac{\partial \boldsymbol{D}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}\boldsymbol{\Psi}\boldsymbol{w}.
\end{aligned}
\tag{3.11}
$$

Let $\boldsymbol{\psi} = \psi(s, a)$, $\overline{\boldsymbol{\psi}} = \sum_{a'} \psi(s', a')$, and $\delta = r(s, a, s') + \gamma \boldsymbol{\theta}^\top \overline{\boldsymbol{\psi}} - \boldsymbol{\theta}^\top \boldsymbol{\psi}$. The first matrix term in the derivative is

$$
\begin{aligned}
\boldsymbol{\Psi}^\top \frac{\partial \boldsymbol{D}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)}(T_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}}) &= \boldsymbol{\Psi}^\top \text{diag}\left\{d(s)\frac{\partial \pi_{\boldsymbol{\theta}}(a|s)}{\partial \boldsymbol{\theta}(i)}\right\}_{(s,a)\in\mathcal{S}\times\mathcal{A}}(\boldsymbol{R} + \gamma \boldsymbol{P}_{\boldsymbol{\theta}}Q_{\boldsymbol{\theta}} - Q_{\boldsymbol{\theta}}) \\
&= \sum_{s,a,s'} d(s)\frac{\partial \pi_{\boldsymbol{\theta}}(a|s)}{\partial \boldsymbol{\theta}(i)}\boldsymbol{\Psi}^\top \delta \\
&= \mathbb{E}\left[\frac{\partial \pi_{\boldsymbol{\theta}}(a|s)/\partial \boldsymbol{\theta}(i)}{\pi_{\boldsymbol{\theta}}(a|s)}\delta\boldsymbol{\psi}^\top\right],
\end{aligned}
\tag{3.12}
$$

where the expectation is over all possible transitions (s, a, s'). Note that δ depends on s'. Let $\boldsymbol{\psi}' = \psi(s', a')$, the second matrix term is

$$
\begin{aligned}
&\boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}} \left(\gamma \frac{\partial \boldsymbol{P}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)} \boldsymbol{\Psi}\boldsymbol{\theta} + \gamma \boldsymbol{P}_{\boldsymbol{\theta}} \boldsymbol{\Psi}(\cdot, i) - \boldsymbol{\Psi}(\cdot, i) \right) \\
&= \boldsymbol{\Psi}^\top \boldsymbol{D}_{\boldsymbol{\theta}} \left(\gamma \begin{bmatrix} \sum_{s',a'} p(s_1, a_1, s') \frac{\partial \pi_{\boldsymbol{\theta}}(a'|s')}{\partial \boldsymbol{\theta}(i)} \boldsymbol{\theta}^\top \boldsymbol{\psi}' \\ \vdots \\ \sum_{s',a'} p(s_n, a_m, s') \frac{\partial \pi_{\boldsymbol{\theta}}(a'|s')}{\partial \boldsymbol{\theta}(i)} \boldsymbol{\theta}^\top \boldsymbol{\psi}' \end{bmatrix} + \gamma \begin{bmatrix} \sum_{s',a'} p(s_1, a_1, s') \pi_{\boldsymbol{\theta}}(a'|s') \boldsymbol{\psi}'(i) \\ \vdots \\ \sum_{s',a'} p(s_n, a_m, s') \pi_{\boldsymbol{\theta}}(a'|s') \boldsymbol{\psi}'(i) \end{bmatrix} - \boldsymbol{\Psi}(\cdot, i) \right) \\
&= \sum_{s,a} d_{\boldsymbol{\theta}}(s, a) \boldsymbol{\psi}^\top \left(\gamma \sum_{s',a'} p(s, a, s') \frac{\partial \pi_{\boldsymbol{\theta}}(a'|s')}{\partial \boldsymbol{\theta}(i)} \boldsymbol{\theta}^\top \boldsymbol{\psi}' + \gamma \sum_{s',a'} p(s, a, s') \pi_{\boldsymbol{\theta}}(a'|s') \boldsymbol{\psi}'(i) - \boldsymbol{\psi}(i) \right) \\
&= \mathbb{E} \left[\left(\gamma \mathbb{E} \left[\frac{\partial \pi_{\boldsymbol{\theta}}(a'|s') / \partial \boldsymbol{\theta}(i)}{\pi_{\boldsymbol{\theta}}(a'|s')} \boldsymbol{\theta}^\top \boldsymbol{\psi}' \right] + \gamma \mathbb{E} \left[\boldsymbol{\psi}'(i) \right] - \boldsymbol{\psi}(i) \right) \boldsymbol{\psi}^\top \right], \qquad (3.13)
\end{aligned}
$$

The third term is

$$
\boldsymbol{w}^\top (\boldsymbol{\Psi}^\top \frac{\partial \boldsymbol{D}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}(i)} \boldsymbol{\Psi}) \boldsymbol{w} = \boldsymbol{w}^\top \left(\sum_{s,a} d(s) \frac{\partial \pi_{\boldsymbol{\theta}}(a|s)}{\partial \boldsymbol{\theta}(i)} \boldsymbol{\psi}\boldsymbol{\psi}^\top \right) \boldsymbol{w} = \boldsymbol{w}^\top \mathbb{E} \left[\frac{\frac{\partial \pi_{\boldsymbol{\theta}}(a|s)}{\partial \boldsymbol{\theta}(i)}}{\pi_{\boldsymbol{\theta}}(a|s)} \boldsymbol{\psi}\boldsymbol{\psi}^\top \right] \boldsymbol{w}. \qquad (3.14)
$$

Using $\frac{\nabla \pi_{\boldsymbol{\theta}}}{\pi_{\boldsymbol{\theta}}} = \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s)}{\pi_{\boldsymbol{\theta}}(a|s)}$ and $\frac{\nabla \pi'_{\boldsymbol{\theta}}}{\pi'_{\boldsymbol{\theta}}} = \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a'|s')}{\pi_{\boldsymbol{\theta}}(a'|s')}$ as a shorthand and assembling the MSPBE gradient gives

$$
\begin{aligned}
&-\frac{1}{2} \nabla_{\boldsymbol{\theta}} \text{MSPBE}(\boldsymbol{\theta}) \\
&= -\left(\mathbb{E} \left[\frac{\nabla \pi_{\boldsymbol{\theta}}}{\pi_{\boldsymbol{\theta}}} \delta \boldsymbol{\psi}'^\top \right] + \mathbb{E} \left[\left(\gamma \mathbb{E} \left[\frac{\nabla \pi'_{\boldsymbol{\theta}}}{\pi'_{\boldsymbol{\theta}}} \boldsymbol{\theta}^\top \boldsymbol{\psi}' \right] + \gamma \mathbb{E} \left[\boldsymbol{\psi}' \right] - \boldsymbol{\psi} \right) \boldsymbol{\psi}^\top \right] \right) \boldsymbol{w} + \frac{1}{2} \boldsymbol{w}^\top \mathbb{E} \left[\frac{\nabla \pi_{\boldsymbol{\theta}}}{\pi_{\boldsymbol{\theta}}} \boldsymbol{\psi}\boldsymbol{\psi}^\top \right] \boldsymbol{w} \\
&= \mathbb{E} \left[\boldsymbol{\psi}\boldsymbol{\psi}^\top \right] \underbrace{\mathbb{E} \left[\boldsymbol{\psi}\boldsymbol{\psi}^\top \right]^{-1} \mathbb{E} \left[\delta \boldsymbol{\psi} \right]}_{=\boldsymbol{w}} - \gamma \mathbb{E} \left[\boldsymbol{\psi}' \boldsymbol{\psi}^\top \right] \boldsymbol{w} - \mathbb{E} \left[\frac{\nabla \pi_{\boldsymbol{\theta}}}{\pi_{\boldsymbol{\theta}}} \delta \boldsymbol{\psi}^\top \right] \boldsymbol{w} - \gamma \mathbb{E} \left[\frac{\nabla \pi'_{\boldsymbol{\theta}}}{\pi'_{\boldsymbol{\theta}}} \boldsymbol{\theta}^\top \boldsymbol{\psi}' \boldsymbol{\psi}^\top \right] \boldsymbol{w} \\
&\qquad + \frac{1}{2} \boldsymbol{w}^\top \mathbb{E} \left[\frac{\nabla \pi_{\boldsymbol{\theta}}}{\pi_{\boldsymbol{\theta}}} \boldsymbol{\psi}\boldsymbol{\psi}^\top \right] \boldsymbol{w} \\
&= \mathbb{E} \left[\delta \boldsymbol{\psi} \right] - \gamma \mathbb{E} \left[\boldsymbol{\psi}' \boldsymbol{\psi}^\top \right] \boldsymbol{w} - \mathbb{E} \left[\frac{\nabla \pi_{\boldsymbol{\theta}}}{\pi_{\boldsymbol{\theta}}} \delta \boldsymbol{\psi}^\top \right] \boldsymbol{w} - \gamma \mathbb{E} \left[\frac{\nabla \pi'_{\boldsymbol{\theta}}}{\pi'_{\boldsymbol{\theta}}} \boldsymbol{\theta}^\top \boldsymbol{\psi}' \boldsymbol{\psi}^\top \right] \boldsymbol{w} + \frac{1}{2} \boldsymbol{w}^\top \mathbb{E} \left[\frac{\nabla \pi_{\boldsymbol{\theta}}}{\pi_{\boldsymbol{\theta}}} \boldsymbol{\psi}\boldsymbol{\psi}^\top \right] \boldsymbol{w}.
\end{aligned} \qquad (3.15)
$$

3.1.1 Off-Policy Conversion

The expectations in (3.15) use the target policy $\pi_{\boldsymbol{\theta}}$ to compute the probabilities of selecting an action. In off-policy training, the transitions (s, a, s') are generated using a different behavior policy b. To be able to derive an off-policy control algorithm generating transitions

using the behavior policy b, we need to re-express the expectations in terms of b. Define

$$\rho \overset{\text{def.}}{=} \frac{\pi_{\boldsymbol{\theta}}(a|s)}{b(a|s)}, \ \rho^{\nabla} \overset{\text{def.}}{=} \frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a|s)}{b(a|s)}, \text{ and } \rho'^{\nabla} \overset{\text{def.}}{=} \frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a'|s')}{b(a'|s')},$$

where ρ is the usual importance sampling ratio. This notation allows us to rewrite (3.15) and (3.10) as

$$\begin{aligned}
&-\frac{1}{2}\nabla_{\boldsymbol{\theta}}\text{MSPBE}(\boldsymbol{\theta}) \\
&\quad = \mathbb{E}_b\left[\rho\delta\boldsymbol{\psi}\right] - \gamma\mathbb{E}_b\left[\rho\boldsymbol{\psi}'\boldsymbol{\psi}^\top\right]\boldsymbol{w} - \mathbb{E}_b\left[\rho^{\nabla}\delta\boldsymbol{\psi}^\top\right]\boldsymbol{w} - \gamma\mathbb{E}_b\left[\rho'^{\nabla}\boldsymbol{\theta}^\top\boldsymbol{\psi}'\boldsymbol{\psi}^\top\right]\boldsymbol{w} + \frac{1}{2}\boldsymbol{w}^\top\mathbb{E}_b\left[\rho^{\nabla}\boldsymbol{\psi}\boldsymbol{\psi}^\top\right]\boldsymbol{w}, \\
&\boldsymbol{w} = \mathbb{E}_b[\rho\boldsymbol{\psi}\boldsymbol{\psi}^\top]^{-1}\mathbb{E}_b[\rho\delta\boldsymbol{\psi}].
\end{aligned} \tag{3.16}$$

The subscript b denotes that the expectations range over all possible transitions where actions are selected with respect to the behaviour policy b.

3.1.2 Sampling the Gradient

To obtain an incremental control algorithm, we derive a stochastic gradient descent method sampling $\nabla_{\boldsymbol{\theta}}\text{MSPBE}(\boldsymbol{\theta})$. Since the auxiliary weight vector $\boldsymbol{w}$ must be estimated independently, the same weight doubling trick presented by Sutton et al. [2009b] is used and the $\boldsymbol{w}_t$ estimate is updated with

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \beta_t\rho(\delta - \boldsymbol{\psi}^\top\boldsymbol{w}_t)\boldsymbol{\psi}, \tag{3.17}$$

where δ is the TD error as defined before. The expectations of (3.16) are sampled by the update rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t\Big[\rho\delta\boldsymbol{\psi} - \rho\gamma\boldsymbol{\psi}'\boldsymbol{\psi}^\top\boldsymbol{w}_t\Big] - \underline{\alpha_t\Big[\rho^{\nabla}\delta\boldsymbol{\psi}^\top\boldsymbol{w}_t + \rho'^{\nabla}\gamma(\theta^\top\boldsymbol{\psi}')(\boldsymbol{\psi}^\top\boldsymbol{w}_t) - \frac{1}{2}\rho^{\nabla}(\boldsymbol{w}_t^\top\boldsymbol{\psi})^2\Big]}. \tag{3.18}$$

This update rule is quite intuitive. It contains the standard GQ(0) update terms minus the underlined correction term in the direction of the policy gradient.

We call the resulting algorithm *Policy-Gradient Q-learning (PGQ)*. A full listing of PGQ is shown in in Algorithm 9. As in all gradient-based TD algorithms, the complexity per time step is linear in the number of basis functions. Similar to Expected SARSA, which we described in Section 2.2, (2.26), we do not sample the action selected by the agent at the

next state s'. Instead we calculate this expectation analytically by computing the expected next state feature vectors $\overline{\boldsymbol{\psi}}$ and $\overline{\boldsymbol{\psi}}^{\nabla}$ over all possible actions before updating the parameter vector $\boldsymbol{\theta}$.

Algorithm 9 PGQ (Policy Gradient Q-learning)

Input: An MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, a target policy $\pi_{\boldsymbol{\theta}}$, a behaviour policy b, a learning rate $\alpha \in (0, 1]$, a start state s_{start}.

Initialize $\boldsymbol{\theta} = 0$, $\boldsymbol{w} = 0$.

$s \leftarrow s_{\text{start}}$, $t \leftarrow 1$

repeat

Sample $b \sim \pi_{\boldsymbol{\theta}}(\cdot|s)$

Take action a, observe reward r and next state s'

$\rho \leftarrow \frac{\pi_{\boldsymbol{\theta}}(a|s)}{b(a|s)}$, $\rho^{\nabla} \leftarrow \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s)}{b(a|s)}$

$\boldsymbol{\psi} \leftarrow \psi(s, a)$, $\overline{\boldsymbol{\psi}} \leftarrow \sum_b \pi_{\boldsymbol{\theta}}(b|s')\psi(s', b)$, $\overline{\boldsymbol{\psi}}^{\nabla} \leftarrow \sum_{a'} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a'|s')\boldsymbol{\theta}^{\top}\boldsymbol{\psi}(s', a')$

$\delta \leftarrow r + \gamma \boldsymbol{\theta}^{\top}\overline{\boldsymbol{\psi}} - \boldsymbol{\theta}^{\top}\boldsymbol{\psi}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_t \left[\rho\delta\boldsymbol{\psi} - \gamma\rho\overline{\boldsymbol{\psi}}(\boldsymbol{\psi}^{\top}\boldsymbol{w}) - \rho^{\nabla}\delta(\boldsymbol{\psi}^{\top}\boldsymbol{w}) - \gamma\overline{\boldsymbol{\psi}}^{\nabla}(\boldsymbol{\psi}^{\top}\boldsymbol{w}) + \frac{1}{2}\rho^{\nabla}(\boldsymbol{w}^{\top}\boldsymbol{\psi})^2\right]$

$\boldsymbol{w} \leftarrow \boldsymbol{w} + \beta_t\rho(\delta - \boldsymbol{\psi}^{\top}\boldsymbol{w})\boldsymbol{\psi}$

$s \leftarrow s'$, $t \leftarrow t + 1$

until state s is terminal

4

Experiments

This chapter presents empirical results comparing the PGQ algorithm to Q-learning and GQ in three standard off-policy control domains: Baird's counterexample, Mountain Car, and Acrobot. Actions are always selected according to a Boltzmann behaviour policy and the GQ and PGQ algorithms both use a Boltzmann target policy. All experiments were repeated with different learning settings and the best performing learning rate setting was chosen for each algorithm and control domain independently.

4.1 Baird Counter Example

The first set of experiments is on the "star" Baird counter example Baird [1995] and compares PGQ to Q-learning and GQ. For this 7 state version, divergence of Q-learning is monotonic and GQ is known to converge [Maei et al., 2010]. The parameter vector $\boldsymbol{\theta}$ corresponding to the action that transitions to the 7th centre state is initialized with $(1, 1, 1, 1, 1, 1, 1, 10)$ and the remaining parameter entries are initialized with 1. The discount factor is set to $\gamma = 0.99$. These experiments differ from Maei et al. [2010] in that the control or target policies are not fixed but use Boltzmann action selection distributions computed directly from the action-value function. Updating is done with dynamic programming sweeps using a uniform stationary distribution $d(s)$, similar to Maei et al..

Figure 4.1 compares the MSPBE and the maximum norm of the $\boldsymbol{Q}$ vector for all state-action pairs for all three algorithms. In the Baird counter example, the rewards are always zero; therefore an optimal action-value estimate is zero and $||\boldsymbol{Q}||_\infty$ should converge to zero. As

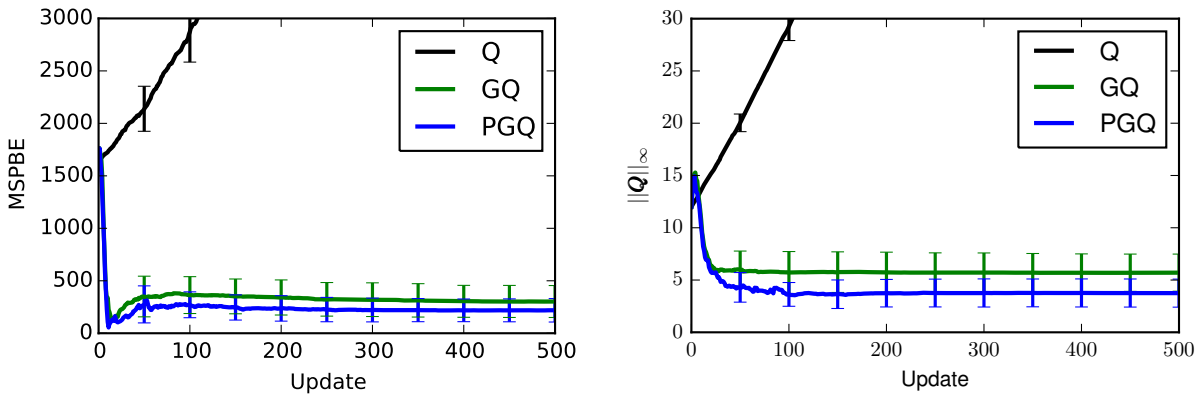

Figure 4.1: MSPBE and maximum of all state-action values for off-policy control averaged over 20 runs. The behaviour policy temperature is $\tau = 10$. The temperature of the target policy used by GQ and PGQ is $\tau = 0.2$. Q-learning uses a learning rate $\alpha = 0.01$ while GQ and PGQ both use $\alpha = 0.1$ and $\beta = 0.1$.

shown before, Q-learning diverges while GQ and PGQ both converge to a low error solution. As can be seen, PGQ finds a solution with lower MSPBE and with lower action-values than the solution found by GQ. These results suggest that PGQ makes more efficient updates and finds a higher quality solution than GQ. However, the updates for both GQ and PGQ still flatten after approximately 150 updates. Further, the plot does not suggest that the action values converge to zero.

4.2 Mountain Car

In the Mountain car task [Sutton and Barto, 1998] the agent drives an underpowered car and the goal is to drive the car out of a valley and up a hill in as few time steps as possible. The action space consists of three actions: accelerate forward, backward, or coast. The state space consists of the position of the car and its velocity. The basis function used for this task computes a bit-vector tiling the two dimensional state space into a 18×18 grid. For each of the three actions, a separate 18×18 grid is used. Figure 4.2 shows the average episode length and the average infinity and L1 norm the vector $\boldsymbol{\theta}$ for each episode under off-policy training using Boltzmann policies. The target temperature for GQ and PGQ is set to $\tau = 0.5$ and the behaviour temperature to $\tau = 1.1$ for all algorithms. Since the

basis function generates a vector with one entry set to one and all others set to zero, the in magnitude largest state-action value equals $||\boldsymbol{\theta}||_\infty$ and $||\boldsymbol{\theta}||_1$ is equal to the sum of all absolute state-action values. First, one can see that the average episode length of GQ increases after approximately 50 episodes, i.e. GQ does not seem to converge to a good solution and instead finds a policy whose performance becomes worse as the number of episodes increases. However, Q-learning and PGQ both find a policy that solves the task faster with Q-learning performing significantly better than PGQ after approximately 150 episodes. The plot of $||\boldsymbol{\theta}||_1$ reveals that all three algorithms seem to converge to different solutions. Q-learning produces a higher average $||\boldsymbol{\theta}||_\infty$ and $||\boldsymbol{\theta}||_1$ than GQ and PGQ which can be contributed to the fact that a higher learning rate of $\alpha = 0.5$ is used.

4.3 Acrobot

The Acrobot task [Sutton and Barto, 1998] consists of a two-link under-actuated robot arm mounted at the top joint. Torque can be applied by the agent in two directions on the middle-joint, or no torque can be applied by the agent. Hence, the action space contains three actions and the state is described by four continuous variables, the angle and angular velocity of the top and middle joint respectively. The basis function is a bit vector tiling the four-dimensional state space into 12 tiles along the two angular positions and 14 tiles along the two angular velocities. Similar to the Mountain car experiment, a separate grid is used for each of the actions. The goal of the agent is to learn how to apply torque to the middle joint in order to move the tip of the arm to a certain height in as few time steps as possible. For this task a control temperature of $\tau = 1.1$ and a target temperature of $\tau = 0.5$ is used. Each episode has a maximum length of 1500 iterations. The experiment is repeated 20 times with each algorithm.

Figure 4.3 compares the average episode length and the average infinity and L1 norm of the vector $\boldsymbol{\theta}$ among Q-learning, GQ, and PGQ. Similarly to the Mountain car experiments, the basis function only generates a bit vector with only one entry set to one and the others to zero, so computing $||\boldsymbol{\theta}||_\infty$ corresponds to finding the in magnitude largest state-action value. The plot of $||\boldsymbol{\theta}||_\infty$ and $||\boldsymbol{\theta}||_1$ in Figure 4.3 show that all three algorithms seem to find

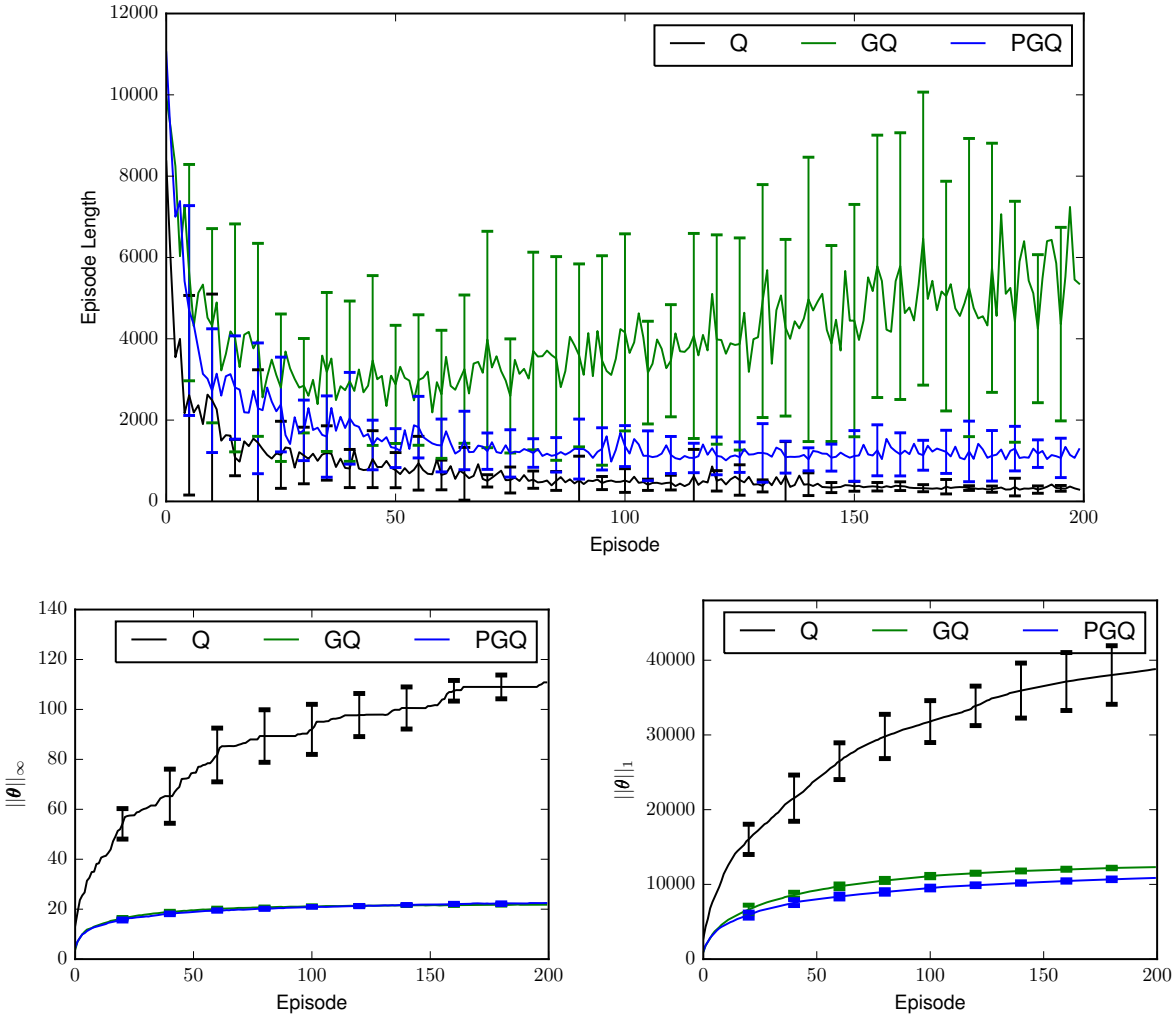

Figure 4.2: Episode length, infinite norm, and L1 norm of the value function parameter $\boldsymbol{\theta}$ on the Mountain car task. Both plots show the same dataset. Each experiment is repeated 20 times. Q-learning uses a learning rate $\alpha = 0.5$ while GQ and PGQ use learning rates $\alpha = 0.1$ and $\beta = 0.005$.

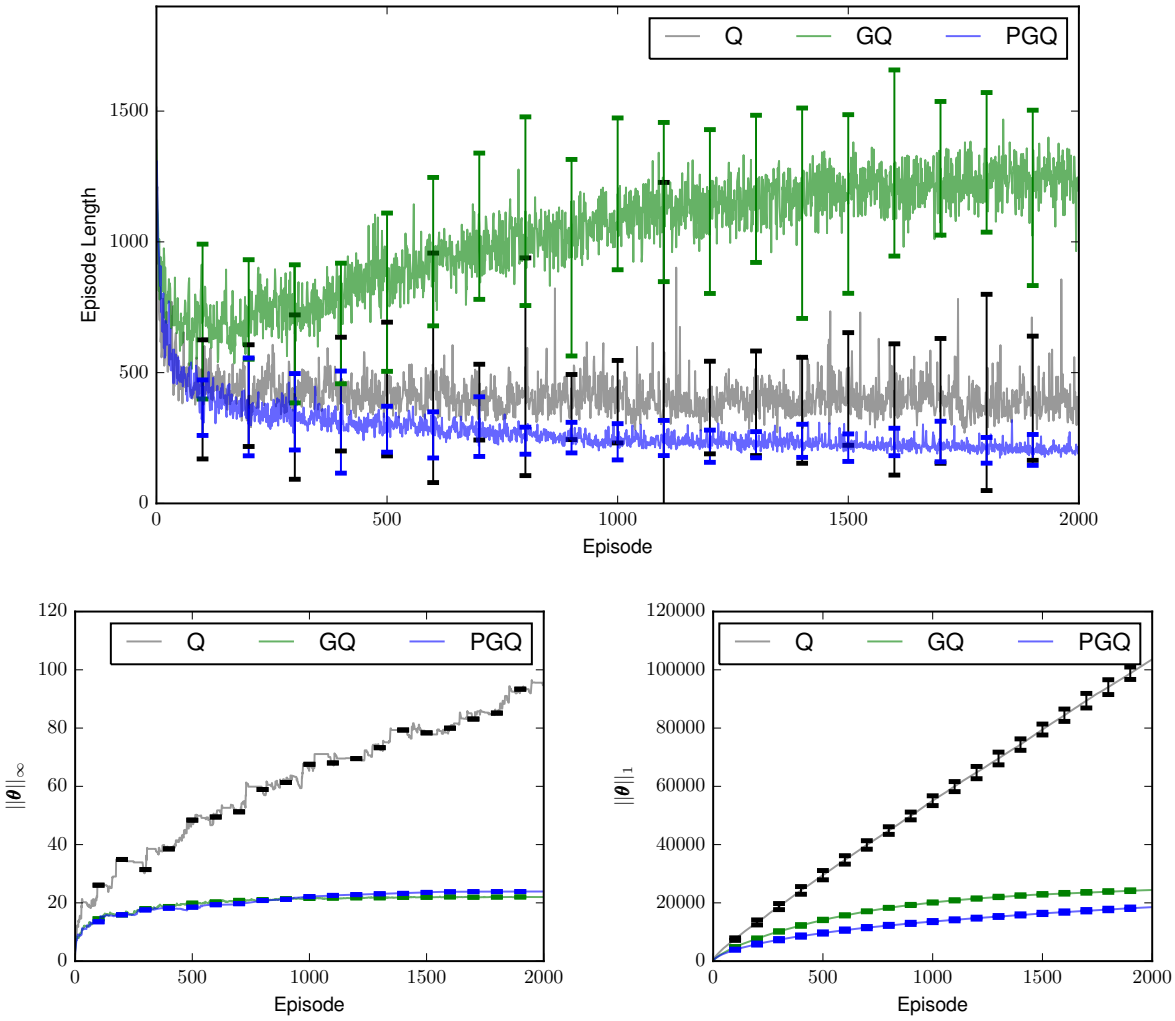

Figure 4.3: Episode length. infinity norm, and L1 norm of the value function parameter $\boldsymbol{\theta}$ on the Acrobot task. Both plots show data from the same experiments and the curves show the averages and standard deviations of 20 repeats. Q-learning uses a learning rate of $\alpha = 0.2$. GQ and PGQ both use $\alpha = 0.1$ and $\beta = 0.005$.

a different solution. In fact, the action values of Q-learning seem to grow linearly without bound as the number of episodes increases. However, Q-learning finds a policy that only performs slightly worse than PGQ in terms of episode length. In comparison to GQ and PGQ, a slow down in the growth rate of the state-action values cannot be observed. PGQ seems to perform best as its average episode length is lower than the average episode of Q-learning and GQ. The average episode length of GQ increases after around 200 episodes which indicates the GQ does not seem to learn an efficient policy. These results highlight the difference between the three algorithms and show that under off-policy training on complex control problems the PGQ algorithm outperforms both Q-learning and GQ.

5

Conclusion and Future Work

5.1 Conclusion

We presented a new gradient based TD-learning algorithm, called PGQ, that uses policy gradients in order to account for changes in the probability with which transitions are sampled. The resulting algorithm is similar to GQ(0) but includes a correction term in the direction of the gradient of the target policy. Our analysis in Chapter 3 accounts for the dependency of the Markov chain induced by a particular policy on the value function parameter vector $\boldsymbol{\theta}$ through the target policy. Finding an optimal control policy is equivalent to finding the policy which induces a Markov chain in the MDP that generates the highest cumulative return. By accounting for the interaction between the value function estimate and the generated Markov chain, our algorithm accounts directly for changes in the probabilities with which actions are sampled. In contrast, existing control algorithms that learn incrementally, such as Q-learning, SARSA, and GQ, do not account for changes in the probabilities of selecting an action. As a result, these methods drift in the space of Markov chains generated by all possible control policies. In fact, SARSA is known to only converge to a specific region in which the value function estimate may oscillate indefinitely [Gordon, 2001, 1996]. Our approach is to view the policy to be a direct operator of the current value function estimate, similar to the improvement operator in Approximate Policy Iteration [Perkins and Precup, 2003]. However, PGQ performs both policy evaluation and policy improvement simultaneously and does not consider them as two separate steps. While we approach the gradient derivation of the MSPBE objective similar to the gradient derivation of the policy gradient algorithm presented

by Sutton et al. [2000], we do not consider the parameters of the policy as a separate vector. Instead, the policy is a function of the value function, i.e. both policy and value function are parametrized by a common parameter vector $\boldsymbol{\theta}$. As a result, a compatibility condition matching the parameters of the policy and value function is not necessary, in contrast to the work presented by Sutton et al. [2000]. Consequently, we do not have to solve a system of equations to find a good approximation of the value function for the current policy estimate. Our method accomplishes both simultaneously.

The experimental results in Chapter 4 indicate that PGQ performs more accurate updates than GQ(0) does under off-policy training because PGQ finds a significantly more efficient policy than GQ(0) does. PGQ is also the only algorithm that seems to remain stable on all tested domains since its action-value estimates always seem to remain bounded. Especially on the Acrobot control domain, the most complex control problem tested, PGQ outperforms both GQ(0) and Q-learning. These results highlight that our more direct approach for searching the space of Markov chains generated by all possible policies results in better performance. Further, the empirical results on the Baird Counter Example in Chapter 4 indicate that PGQ also converges under off-policy control with linear function approximation. The GQ(0) algorithm also seems to converge on the Baird Counter example in the control case. However, a convergence proof for both methods for the control case is still unknown.

5.2 Future Work

Further analyzing PGQ in order to proof convergence in the off-policy control case would be a natrual continuation of this research. The ODE method [Borkar and Meyn, 1999], which was used to proof convergence of TDC, does not directly extend to the control case as this proof technique assumes transition data to be i.i.d. which is not the case in the control context. One may be able to leverage ideas from the convergence proof of SARSA [Singh et al., 2000] or the Approximate Policy Iteration algorithm [Perkins and Precup, 2003]. However, these proofs only consider the on-policy control case.

Another interesting continuation of this research would be to extend this method to the non-linear function approximation case. The presented approach is tied to linear function

approximation solely through the use of the projection operator Π in the MSPBE objective. Intuitively, this operator performs a fit of the linear objective function to minimize the TD-errors at every state-action pair, similar to linear regression. Maei et al. [2009] present a gradient based TD algorithm which uses a projection operator for non-linear value functions that are Lipschitz continuous. One may be able to merge their results with the approach presented in this thesis to derive a control algorithm that can be used with non-linear function approximation.

Investigating how to account for the interaction between the stationary distribution $d_t(s)$ and changes in the parameter vector $\boldsymbol{\theta}$ may give rise to a better off-policy control algorithm. While it is valid to assume the overall stationary distribution $d(s)$ to be steady, being able to account for changes in the distribution $d_t(s)$ on a per time step basis would allow the algorithm to model more closely the process with which transitions are generated. Consequently, the algorithm may use the transition data more efficiently and converge faster.

Bibliography

Leemon Baird. Residual Algorithms: Reinforcement Learning with Function Approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37. Morgan Kaufmann, 1995.

Dimitri P. Bertsekas. *Neuro-Dynamic Programming*. Athena Scientific, 1 edition, May 1 1996.

Dimitri P. Bertsekas. Approximate policy iteration: A survey and some new methods. *Journal of Control Theory and Applications*, pages 310–335, June 2011.

V. S. Borkar and S.P. Meyn. The o.d.e. method for convergence of stochastic approximation and reinforcement learning. *SIAM J. Control Optim*, 38:447–469, 1999.

Vivek S. Borkar. Stochastic approximation with two time scales. *Systems & Control Letters*, 29(5):291 – 294, 1997.

Thomas Degris, Martha White, and Richard Sutton. Off-Policy Actor-Critic. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, pages 457–464, New York, NY, USA, July 2012. Omnipress.

Geoffrey J. Gordon. Chattering in SARSA(λ) - A CMU Learning Lab Internal Report. Technical report, Carnegie Mellon University, 1996.

Geoffrey J. Gordon. Reinforcement Learning with Function Approximation Converges to a Region. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, pages 1040–1046. The MIT Press, 2001.

Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. Convergence of stochastic iterative dynamic programming algorithms. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 703–710. Morgan-Kaufmann, 1994.

H. R. Maei and R. S. Sutton. GQ(λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the Third Conference on*

Artificial General Intelligence, Advances in Intelligent Systems Research. Atlantis Press, March 2010.

Hamid R. Maei, Csaba Szepesvári, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1204–1212. Curran Associates, Inc., 2009.

Hamid Reza Maei, Csaba Szepesvari, Shalabh Bhatnagar, and Richard S. Sutton. Toward Off-Policy Learning Control with Function Approximation. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 719–726, Haifa, Israel, June 2010. Omnipress.

Andrew Ng and Stuart Russell. Algorithms for Inverse Reinforcement Learning. In Jorge Pinho De Sousa, editor, *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 663–670. Morgan Kaufmann Publishers Inc., 2000.

Theodore J. Perkins and Doina Precup. A convergent form of approximate policy iteration. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1627–1634. MIT Press, 2003.

Doina Precup, Richard S. Sutton, and Sanjoy Dasgupta. Off-policy temporal difference learning with function approximation. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 417–424, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

Gavin Adrian Rummery. *Problem Solving with Reinforcment Learning*. PhD thesis, Cambridge University Engineering Department, Trumpington Street, Cambridge, CD2 1PZ, England, 1995.

Satinder Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. In *Machine Learning*, volume 39, pages 287–308. Kluwer Academic Publishers, February 2000.

Richard Sutton, Hamid Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvari, and Eric Wiewiora. Fast Gradient-Descent Methods for Temporal-Difference Learning with Linear Function Approximation. In Léon Bottou and Michael Littman, editors, *Proceedings of the 26th International Conference on Machine Learning*, pages 993–1000, Montreal, June 2009a. Omnipress.

Richard S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1):9–44, August 1988.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book. MIT Press, Cambridge, MA, 1 edition, 1998.

Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

Richard S Sutton, Hamid R. Maei, and Csaba Szepesvári. A Convergent $O(n)$ Temporal-difference Algorithm for Off-policy Learning with Linear Function Approximation. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1609–1616. Curran Associates, Inc., 2009b.

Philip Thomas. Bias in natural actor-critic algorithms. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 441–448. JMLR Workshop and Conference Proceedings, 2014.

Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

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