Politecnico di Milano
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering

Analyzing the Complexity of Mean-Volatility Algorithms for Risk-Averse Reinforcement Learning

Advisor: Prof. Marcello Restelli
Co-Advisor: Dott. Lorenzo Bisi

Thesis by:
Khaled Eldowa Matr. 940070

Academic Year 2020–2021

Abstract

The goal in the standard Reinforcement Learning problem is to find a policy that optimizes the expected accumulated reward collected while interacting with the environment. Such an objective, however, is not enough in a lot of real-life applications, like finance, where controlling the uncertainty of the outcome is imperative. The mean-volatility objective penalizes, through a tunable parameter, policies with high variance of the per-step reward. Unlike most risk-averse objectives, which consider some statistical property of the accumulated reward, the mean-volatility objective admits simple linear Bellman equations that resemble, up to a reward transformation, those of the risk-neutral case. This potentially enables adapting the large wealth of algorithms and results from the standard literature to the risk-averse case. However, the difficulty with the aforementioned reward transformation is that it requires knowing the expected accumulated reward of the policy in question, a quantity that is usually unknown a priori. This work focuses on policy evaluation and optimization algorithms for the mean-volatility objective. More specifically, this thesis focuses on analyzing the sample complexity of such algorithms compared to their risk-neutral counterparts, especially in problems with large or continuous state and action spaces. We consider two different approaches for policy evaluation: the direct method and the factored method. Moreover, we analyze a full actor-critic algorithm adapted for the mean-volatility objective. Experiments are then carried out to test these algorithms in a simple environment that exhibits some trade-off between optimality, in expectation, and uncertainty of outcome.

Keywords: Reinforcement Learning, Risk Aversion, Reward Volatility, Actor-Critic, Finite-Sample Analysis.

Contents

List of Figures

List of Algorithms

Introduction

The field of reinforcement learning [1] is concerned with how agents can learn to act optimally in an unfamiliar environment via trial and error, aiming to maximize the accumulated reward collected when interacting with the environment. Reinforcement learning has enjoyed a lot of success in recent years; some illustrious achievements include mastering the game of Go [2] and playing Atari games [3]. Nonetheless, when moving from the realm of games to the real world, reinforcement learning faces a number of extra challenges. One being the poor sample efficiency of a lot of reinforcement learning methods, which can lead to prohibitively large costs in real-life applications. Another problem is ensuring the safety of the agent and the environment during the training phase, which is challenging mainly due to the exploratory nature of reinforcement learning. Another crucial issue, which is the focus of this thesis, is the need to control the risk resulting from the inherit stochasticity of the environment. In a lot of real-life applications, like training an automatic trading agent or an autonomous car, acting optimally in expectation, which is the standard objective in reinforcement learning, is not enough without having safety guarantees, specially against rare but catastrophic events.

Risk aversion has received some attention in the reinforcement learning literature. One approach is to learn to avoid states linked to bad or catastrophic events [4], but the prevailing paradigm is to modify the objective of the problem to include a notion that quantifies risk. Examples of this include using the variance of the accumulated reward [5] or its Conditional Value at Risk (CVaR) [6]. The main difficulty with modifying the objective is that the standard tools of reinforcement learning are usually not applicable anymore, and ad-hoc methods have to be devised. More recently, a new notion of risk, dubbed the reward-volatility, has been proposed in [7]. The idea is to consider the variance of the step-reward instead of the accumulated reward. This objective leads to a simpler setting where adapting the standard risk-neutral tools is easier, while still serving as a meaningful proxy for the variance of the accumulated reward.

Contributions

The contributions of this thesis are primarily theoretical. We focus on the mean-volatility objective [7], which provides a trade-off, controlled via a tunable parameter, between minimizing the reward-volatility and maximizing the expected value of the accumulated reward (also called the expected return). As we will see later, standard reinforcement learning methods can be adapted for this new objective by transforming the step-rewards. The required transformation, however, depends on the adopted policy (the way of acting in the environment) as it requires the knowledge of its expected return, which usually needs to be estimated. The aim of this thesis is to understand the cost incurred by having to estimate the expected return and the effect of its inaccuracy on the performance of the algorithms. More specifically, we consider two approaches for performing policy evaluation under the mean-volatility objective, and analyze their sample complexity. The first of these approaches is called the direct method, in which we attempt to adapt any specific policy evaluation algorithm to the mean-volatility objective using the aforementioned reward transformation. The other approach, called the factored method, circumvents the need for performing a costly reward transformation; it can use any risk-neutral policy evaluation algorithm in an off-the-shelf manner as one part of the algorithm. Moreover, we perform finite-sample analysis of a complete actor-critic algorithm adapted for the mean-volatility objective, and we compare it with its risk-neutral counterpart.

Outline

The thesis is structured as follows:

- We start in Chapter 1, by a brief overview over the basic notions and algorithms of reinforcement learning.
- In Chapter 2, we provide an overview of three (more) advanced topics in reinforcement learning. The first being policy evaluation using function approximation, the second is policy gradient methods, and third is the problem of risk-aversion.
- We then provide the problem formulation in Chapter 3, and introduce the main methods to be analyzed in the rest of the thesis.
- In Chapter 4, we focus on the factored method, and provide a general scheme for obtaining error bounds on the learned transformed value function.
- In Chapter 5, we provide finite sample analysis for a full actor-critic algorithm where the critic uses the direct method. We also compare the performance of both the critic and the actor with their risk-neutral counterparts.
- Lastly, in Chapter 6, we carry out simple experiments to assess the soundness and performance of the considered methods.

Chapter 1

Reinforcement Learning I

Reinforcement learning (RL) is one of the three main sub-fields of machine learning alongside supervised and unsupervised learning. It is concerned with the problem of an agent learning to act optimally (in some sense) in a sequential decision making problem in an environment that is usually stochastic and whose workings are generally not fully known. Such an agent, thus, has to learn about the quality of the possible decisions by interacting with the environment in a trial and error fashion. In this way, reinforcement learning, out of all the flavors of artificial intelligence, is seemingly the one closest in nature to the way that animals learn in the real world.

A reinforcement learning problem has three main components: the states, the actions, and the goal. A state is a description of the environment at a certain time; it is the context in which the agent makes its decisions. Taking actions is the way through which the agent can affect the state of the environment. And naturally, the agent has a goal related to the preferable way that the environment should be throughout the interaction period. Goals in reinforcement learning are modelled via a reward function. Roughly speaking, after every action taken by the agent in some state, a reward (a numerical value) is presented to the agent. The agent's goal is then to accumulate as much reward as possible over the course of its interaction with the environment. A crucial thing to point out here is that the quality of an action is not determined solely by the immediate reward received after executing it, but it also depends on the quality of the resulting state of the environment in terms of the possible sequences of rewards that can be collected starting from that state.

In this chapter, the basic algorithms for reinforcement learning are briefly discussed. First, Markov decision processes (MDPs) are introduced. They are the formal way that a reinforcement learning problem is modelled. Second, a brief overview of dynamic programming methods is provided. These are methods for achieving optimal control provided that the model of the environment is known. Lastly, we see how these methods can be extended to the case where the environment's model is not available and learning by interacting with the environment is necessary. In this chapter we assume that the state and action spaces are finite and small enough that the methods presented are tractable. Methods that are suitable for dealing with large or infinite state and action spaces are presented in the next chapter. Unless otherwise stated, all the results mentioned in this chapter are cited from [1].

1.1 Markov Decision Processes

A discrete-time MDP models a sequential decision making problem where at each time step, an action is chosen and executed by an agent and accordingly a numerical reward is revealed and the system (usually stochastically) transitions to a new state. A typical Markov decision process can be represented by the quintuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$. Here $\mathcal{S}$ is the (finite) set of possible states, and $\mathcal{A}$ is the (finite) set of possible actions. $P : \mathcal{S} \times \mathcal{A} \to \mathbb{P}(\mathcal{S})$ defines a transition kernel that maps pairs of states and actions to probability distributions over the possible next states, and $P(s_{t+1} = s' | s_t = s, a_t = a)$ gives the probability of transitioning to state s' in time step $t+1$ after executing action a in state s at time step t. Naturally, we make use of the assumption that the process is Markovian, which means that $P(s_{t+1} = s' | s_t = s, a_t = a, s_{t-1} = x_{t-1}, a_{t-1} = y_{t-1}, ..., s_0 = x_0, a_0 = y_0) = P(s_{t+1} = s' | s_t = s, a_t = a)$. In other words, knowing the last state and action, the distribution over the possible next states is independent of the rest of the history. $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function mapping pairs of states and actions to a real number. The reward function can be stochastic, but it is assumed here to be deterministic for simplicity. Also, R_t is taken to be the reward received in time t after executing action a_t in state s_t. Finally, γ is the discount factor; it describes how much we value future rewards compared to immediate rewards, but more on it in a bit.

We can distinguish between two types of tasks: *episodic* tasks and *continuing* tasks. Episodic tasks break naturally into sequences where the end of one does not affect how the next starts. It could be that the task ends after a fixed number of times steps, or there maybe a special terminal state upon reaching which the episode ends. In continuing tasks, on the other hand, the agent interacts with the environment indefinitely. Episodic tasks can be seen as continuing tasks in which upon reaching a terminal state, the environment gets stuck there indefinitely receiving a reward of zero regardless of the executed action. A relevant quantity to introduce is the (discounted) return:

$$G_t := \sum_{k=0}^{\infty} \gamma^k R_{t+k}. \tag{1.1}$$

It is the discounted sum of rewards received starting from time t; it is one way of describing the accumulated rewards from that time on. The discount factor γ can take values in the interval $[0, 1]$, unless the task is continuing, in which case it cannot be 1. In this way, for continuing tasks, the return is guaranteed to be finite as long as the rewards are bounded in absolute value. Discounting the rewards thus provides a mathematically convenient way of representing the accumulated rewards in the long run. But, as mentioned before, it also signifies how much we value future rewards compared to immediate ones, which can make it quite a relevant quantity in some applications.

A policy describes a way of acting in the environment. In its most general notion, a policy maps the history of states encountered so far to a probability distribution over the actions to be executed in the present. An interesting class of policies (which we will focus on) is the class of stationary (Markovian) policies that map the current state to a distribution over the actions regardless of the history encountered so far and regardless of which time steps we are at. Thus, a stationary policy is a mapping $\pi : \mathcal{S} \to \mathbb{P}(\mathcal{A})$, and we denote by $\pi(a|s)$ the probability of executing action a at state s. As a special case, a deterministic stationary policy maps states to actions in a deterministic way. In this case we (by overloading the notation) denote by $\pi(s)$ the action that policy π chooses at state s. Our ultimate goal is to find an optimal policy, but we need to first define a notion of optimality. In other words, we need a way to compare and order policies.

Suppose we adopt a certain policy π and we want to evaluate the quality of a certain action that the policy chose in a certain state. We know that it is not enough to judge it by the immediate reward received, we also need to assess the quality of the state that it has taken us to. A natural way to define the value of a state (when executing a certain policy π) is via the *state-value function* $V^\pi : \mathcal{S} \to \mathbb{R}$ defined as follows:

$$V^\pi(s) := \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k}\middle| s_t = s\right]. \tag{1.2}$$

That is, the expected return when starting from state s and executing policy π onward. It is easy to see that the value function conveniently admits a form of the Bellman equation:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[R_t|s_t = s] + \gamma\mathbb{E}_\pi[V^\pi(s_{t+1})|s_t = s] \qquad (1.3)\\ &= \sum_{a\in\mathcal{A}} \pi(a|s)R(s,a) + \gamma\sum_{a\in\mathcal{A}}\pi(a|s)\sum_{s'\in\mathcal{S}} P(s'|s,a)V^\pi(s') \\ &= \sum_{a\in\mathcal{A}} \pi(a|s)R(s,a) + \gamma\sum_{s'\in\mathcal{S}} P^\pi(s'|s)V^\pi(s'), \end{aligned}$$

where $P^\pi(s'|s)$ is the probability of a one-step transition from state s to state s' when executing policy π. The Bellman equation above is quite vital for dynamic programming (and reinforcement learning) algorithms; it asserts that the value of a state is the expected value of the immediate reward plus the expected value of the value function at the next state. Analogously, we can evaluate state-action pairs (when executing a policy π) using the action value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defined as follows:

$$Q^\pi(s,a) := \mathbb{E}_\pi[G_t|s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k}\middle| s_t = s, a_t = a\right].$$

Like the state-value function, the action-value function admits a Bellman equation:

$$\begin{aligned} Q^\pi(s,a) &= R(s,a) + \gamma\,\mathbb{E}[V^\pi(s_{t+1})|s_t = s, a_t = a] \\ &= R(s,a) + \gamma\sum_{s'\in\mathcal{S}} P(s'|s,a)V^\pi(s') \qquad (1.4) \end{aligned}$$

$$= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^{\pi}(s',a').$$

We can use the state-value function to define a partial order over policies. Specifically, for a pair of stationary policies π and π', we can say that $\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)\ \forall s \in \mathcal{S}$. With this notion, we can search for an optimal policy π^* in the sense that $V^{\pi^*}(s) = \max_{\pi} V^{\pi^*}(s)\ \forall s \in \mathcal{S}$. Fortunately, at least one such policy exists. For any such policy, we denote its state-value and action-value functions as V^* and Q^* respectively. Note that given Q^*, we can find an optimal policy by constructing a policy that is greedy with respect to Q^*, i.e. $\pi^*(a|s) > 0 \implies Q^*(s,a) = \max_{a' \in \mathcal{A}} Q^*(s,a')$. Such a policy can be made deterministic by choosing a single optimal action at each state. This then means that there always exists a deterministic stationary policy that is optimal. We can also find an optimal policy given only V^* and the reward function, but it requires performing a one-step search (using (1.4)) at each state to determine the optimal actions.

The Bellman equation for V^* can be written in a special way:

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}} Q^*(s,a) \\ &= \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \right]. \end{aligned}$$

And similarly for Q^*:

$$\begin{aligned} Q^*(s,a) &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \\ &= R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{a' \in \mathcal{A}} Q^*(s',a'). \end{aligned}$$

Any of these two Bellman (optimality) equations define a system of non-linear equations that in principle, and with perfect knowledge of the environment, can be solved to obtain the (unique) optimal value function, and consequently, an optimal policy. However, solving such systems in closed form is generally not possible, and an exhaustive search is prohibitively expensive. One alternative is using dynamic programming methods. These are iterative methods for solving Bellman equations more efficiently.

1.2 Dynamic Programming

In this section, dynamic programming methods for both policy evaluation and finding an optimal policy are briefly described. Note that in this section, a perfect model of the environment is assumed to be available.

1.2.1 Iterative Policy Evaluation

Let's for now consider the problem of policy evaluation: given a policy π, we would like to find its value function V^{π}. Note that since we are working in a finite state space, a value function can be seen as a function from the state space to the real line or a vector of size $|\mathcal{S}|$ (The cardinality of $\mathcal{S}$) where each component denotes the value of a state. Now define the Bellman operator $T^\pi : \mathbb{R}^{|\mathcal{S}|} \rightarrow \mathbb{R}^{|\mathcal{S}|}$ such that:

$$(T^\pi f)(s) = \sum_{a \in \mathcal{A}} \pi(a|s)R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P^\pi(s'|s)f(s'), \tag{1.5}$$

where f is a generic value function that does not necessarily correspond to an actual policy. Also, define $(T^\pi)^k f$ as the recursive application of T^π k times staring from f, and $\|f\|_\infty = \max_{s \in \mathcal{S}} |f(s)|$ as the supremum norm of a value function f. T^π enjoys some interesting properties, some of which are summarized in the next proposition.

Proposition 1.1. *[8] The Bellman operator T^π defined above has the following properties:*

(i) Monotonicity. *For any value functions f and f' such that $f(s) \geq f'(s)$ $\forall s \in \mathcal{S}$, then*

$$(T^\pi f)(s) \geq (T^\pi f')(s) \; \forall s \in \mathcal{S}.$$

(ii) Contraction. *For any value functions f and f', we have that*

$$\|T^\pi f - T^\pi f'\|_\infty \leq \gamma \|f - f'\|_\infty \; \forall s \in \mathcal{S}.$$

(iii) *For any value function f,*

$$\lim_{k \rightarrow \infty} (T^\pi)^k f = V^\pi.$$

Moreover, $T^\pi V^\pi = V^\pi$ and V^π is the only solution to this equation.

Motivated by these properties, we can introduce the *iterative policy evaluation* algorithm. In this algorithm, we start from an arbitrary value function f_0 and obtain a sequence of value functions $\{f_k\}$ by repeatedly applying T^π, i.e. $f_{k+1} = T^\pi f_k$. This sequence converges in the limit to the true value function V^π, but in practice, the algorithm is stopped when the iterations cease to produce significant (according to some threshold) change. Note that a similar operator (and algorithm) can also be defined for the action-value function Q^π.

1.2.2 Value Iteration

Shifting focus to the problem of finding an optimal policy (or its value function), we can define an analogous operator $T^* : \mathbb{R}^{|\mathcal{S}|} \rightarrow \mathbb{R}^{|\mathcal{S}|}$ such that:

$$(T^* f)(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)f(s') \right],$$

where f is again a generic value function that does not necessarily correspond to an actual policy. Some of the key properties of T^* are summarized in the following proposition.

Proposition 1.2. *[8] The Bellman operator T^* defined above has the following properties:*

(i) Monotonicity. *For any value functions f and f' such that $f(s) \geq f'(s)$ $\forall s \in \mathcal{S}$, then*

$$(T^* f)(s) \geq (T^* f')(s) \, \forall s \in \mathcal{S}.$$

(ii) Contraction. *For any value functions f and f', we have that*

$$\|T^* f - T^* f'\|_\infty \leq \gamma \|f - f'\|_\infty \, \forall s \in \mathcal{S}.$$

(iii) *For any value function f,*

$$\lim_{k \longrightarrow \infty} (T^*)^k f = V^*.$$

Moreover, $T^* V^* = V^*$ and V^* is the only solution to this equation.

(iv) *Any stationary policy π is optimal if and only if*

$$T^\pi V^* = T^* V^* = V^*.$$

With these properties in mind, we can describe the *value iteration* algorithm. Starting from an arbitrary value function f_0, we produce a sequence of value functions $\{f_k\}$ by repeatedly applying T^*, i.e. $f_{k+1} = T^* f_k$. This sequence converges in the limit to the optimal value function V^*

1.2.3 Policy Iteration

Suppose we have a (deterministic) policy π whose value function V^π (or Q^π) is available, we would like to use this information to construct an improved policy π'. One way to do this is by selecting π' as a greedy policy with respect to V^π (or Q^π). That is, $\pi'(s) = \arg\max_{a \in \mathcal{A}} Q^\pi(s, a) \; \forall s \in \mathcal{S}$, where V^π can be used instead of Q^π at the cost of performing a one-step search. The following policy improvement theorem asserts this claim.

Theorem 1.1. *[1] For a pair of deterministic policies π and π' such that*

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s) \, \forall s \in \mathcal{S},$$

we have that

$$V^{\pi'}(s) \geq V^\pi(s) \, \forall s \in \mathcal{S}.$$

This result can also be generalized to stochastic policies. Note that the condition of the theorem is satisfied if π' is a greedy policy as described before. More importantly, when there is no improvement, i.e. $V^{\pi'}(s) = V^\pi(s) \, \forall s \in \mathcal{S}$, then it can be shown that V^π satisfies the Bellman optimality equations, which makes it an optimal policy.

The *policy iteration* algorithm makes use of these properties. It consists of two interleaved stages applied repeatedly until we reach an optimal policy. The first stage is policy evaluation: given a policy π_k, we determine its value function V^{π_k}. The second stage is policy improvement: given V^{π_k}, we determine π_{k+1} that is greedy with respect to V^{π_k}, and thus is guaranteed to be at least as good as V^{π_k}. These two stages are then repeated starting from the new policy, and so on until the policy improvement step fails to improve the policy, which means that we have reached an optimal policy. The produced sequence of policies convergence in a finite number of steps to an optimal policy since our MDP admits a finite number of deterministic policies. If we use iterative policy evaluation to find the value function, it might seem like an issue that it only converges in the limit to the true value function. However, we need not obtain the exact true value function at every iteration in order for the algorithm to converge to an optimal policy. In fact, it's sufficient that our value function estimate is moved towards the true one, and thus the policy evaluation step can be truncated. One extreme way of doing that is by applying only a single sweep of the iterative policy evaluation algorithm between the policy improvement steps. Interestingly, this reduces the algorithm back to value iteration. However, interleaving policy improvement steps with multiple sweeps of the iterative policy evaluation algorithm can usually lead to faster convergence of the (generalized) policy iteration algorithm.

1.3 Model-Free Prediction

We now return to the standard reinforcement learning setting in which the model of the environment is not available but a simulator is. Model-free methods, as opposed to model-based ones, do not use or attempt to learn an explicit model of the environment. Instead, they try to solve the problem directly using data obtained while interacting with the environment. In this section, we consider methods for model-free prediction, i.e. policy evaluation. While in the next one, methods for model-free control, i.e. attempting to approximate an optimal policy, are presented.

1.3.1 Monte-Carlo Methods

Given a stationary policy π, armed with a simulator, and tasked with finding $V^{\pi}(s)$ for a certain state s, the simplest idea that comes to one's mind is obtaining a number of unbiased samples of the return starting from that state and averaging them. This is particularly simple for episodic tasks; one can simulate a number of episodes while acting with π, and for each encountered state, we can use the return starting from that point till the end of the episode as an unbiased sample of the value function of that state. For each state, these samples are independent as long as only the return from the first visit to the state in each episode is used. This algorithm is known as the *first-visit* algorithm as opposed to the *every-visit* algorithm where the return is used as a sample on every visit to the state. For the first-visit algorithm, the estimated value of each state is an empirical mean of i.i.d. (independent and identically distributed) realizations of the true value function, and thus converges with probability 1 to the true value function in the limit of infinite samples by virtue of the strong law of large numbers. The every-visit version can also be shown to converge to the true value function.

The discussed Monte-Carlo methods are certainly sound, but they have a number of shortcomings. They are not immediately adaptable to continuing problems were the notion of an episode is not naturally defined. They are not truly on-line methods in the sense that they only learn from complete episodes, thus they do not learn concurrently with the sampling process. Also, the return can have a high variance, specially for long trajectories, which can lead to slow convergence. The temporal difference methods described next use biased samples of the return, but overcome the mentioned shortcomings of the Monte-Carlo methods.

1.3.2 Temporal Difference Methods

To put temporal difference methods in context, we can start from a convenient modification of the every-visit Monte-Carlo algorithms. Denote by V our (running) estimate of V^π, and fix a state $s \in \mathcal{S}$. Suppose that $N(s)$ denotes the number of times we have visited state s so far. This means that we have $N(s)$ samples of the return starting from s, which we denote by G_i^s for $i = 1, ..., N(s)$. The every-visit Monte-Carlo algorithm would then determine V as the sample average of these returns:

$$V(s) = \frac{\sum_{i=1}^{N(s)} G_i^s}{N(s)}.$$

Instead of using (and storing) the whole batch, the sample average can be calculated incrementally and updated for each state as soon as it is visited again. Suppose that for every $s \in \mathcal{S}$, $V(s)$ (initialized as zero) is kept as the average of the samples of the return starting from s that have been encountered so far. Now, suppose we have simulated a new episode, we can update $V(s_t)$ (where s_t is the state at time-step t) as follows:

$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s(t))}(G_t - V(s_t)),$$

where $N(s)$ is incremented by one on every new visit to s. A more general version of the previous update rule is:

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t)), \tag{1.6}$$

where α is a step-size parameter that controls how much we "forget" old samples, which is of particular relevance in non-stationary problems. The update rule (1.6) resembles a Robbins-Monro stochastic approximation method for solving the equations: (the dependence on π is kept implicit)

$$V(s) = \mathbb{E}[G_t | s_t = s]$$

for the unknowns $V(s)$ for every state s [8]. Inspired by the Bellman equation for V^π, we can use $\mathbb{E}[R_t + \gamma V^\pi(s_{t+1}) | s_t = s]$ instead of $\mathbb{E}[G_t | s_t = s]$. However, we cannot obtain unbiased samples of $\mathbb{E}[R_t + \gamma V^\pi(s_{t+1})]$ to use as targets in (1.6) instead of G_t since we do not know V^π. One idea is to use our running estimate V instead of V^π in the targets. This gives rise to the following *TD(0)* update rule:

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t + \gamma V(s_{t+1}) - V(s_t)). \tag{1.7}$$

Such use of an estimate to learn another estimate is called *bootstrapping*, and it's one of the distinguishing features of temporal difference methods. We can also see that we do not need to wait for episodes to end (the notion of an episode is actually irrelevant here) to update our estimate; we can learn in a truly on-line and incremental fashion. And although TD(0)'s targets are biased, they usually exhibit lower variance compared to Monte-Carlo targets as they usually depend on a smaller number of random events. TD(0) has been proven to converge to the true value function with probability 1 if the step-size parameter, *for each state*, is reduced along the iterations according to the following standard stochastic approximation conditions:

Condition 1.1. *The sequence* $\{\alpha_t\}$ *satisfies:*

- $\sum_{t=0}^{\infty} \alpha_t = \infty$.
- $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$.

Despite its use of bootstrapping, TD(0) is still a sound algorithm. In fact, it is known to be usually more efficient than Monte-Carlo methods in practice. TD(0) is the simplest temporal difference algorithm. There are other temporal difference algorithms like n-step TD or TD(λ) that bridge the gap between TD(0) and Monte-Carlo algorithms.

1.4 Model-Free Control

Now that we have discussed methods for policy evaluation that do not require a model of the environment, we can attempt to use these methods to extend the general policy iteration framework to the model-free case. Since we rely on interacting with the environment to learn in the model-free control problem, we need to distinguish between two types of model-free control algorithms: *on-policy* control and *off-policy* control. In on-policy algorithms, the policy that is used to interact with the environment is the same as the one that is being evaluated or improved. Whereas in off-policy methods, we are evaluating or improving a different policy than the one used to collect the data. In the following, we briefly discuss a classic example of each type: SARSA for on-policy control and Q-learning for off-policy control.

1.4.1 SARSA

When attempting to use the policy iteration framework in the model-free setting, we face some immediate challenges. The first is that obtaining a greedy policy with respect to a state-value function requires the knowledge of the environment's model. This, however, can be circumvented by learning the action-value function instead of the state-value function in the policy evaluation step. A more serious challenge concerns the issue of sufficient exploration. If we do not encounter a certain state (or execute a certain action in a certain state) often enough, we will not have enough information about it and will not be able judge its value. To be able to properly evaluate (and consequently improve) a policy, we need to ensure sufficient exploration of the state and action spaces. This issue is most serious for deterministic greedy policies in the on-policy case since at each state, all but one action are never chosen, and thus we will not be able to ever learn their value. The off-policy control paradigm offers a natural way to deal with this issue. However, in the on-policy case, we need a way to ensure sufficient exploration. One such way is using ϵ-greedy policies instead of purely greedy ones. An ϵ-greedy policy with respect to an action value function Q selects, at any state, the greedy action with probability $1-\epsilon$ and a random action with probability ϵ. In other words,

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{|\mathcal{A}|} + 1 - \epsilon & \text{if } a = \arg\max_{a' \in \mathcal{A}} Q(s, a') \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases}$$

for every state s. An ϵ-greedy policy with respect to Q is said to be an ϵ-soft version of the greedy policy with respect to Q. With an argument analogous to theorem (1.1), it can be shown that for an ϵ-greedy policy π' with respect to Q^π, $V^{\pi'}(s) \geq V^\pi(s)\ \forall s \in \mathcal{S}$ for any ϵ-soft policy π. Moreover, if we fail to obtain any improvement, then π is an optimal policy among ϵ-soft policies.

The SARSA algorithm is a simple algorithm for on-policy control. It applies the framework of generalized policy iteration (where we alternate between partial policy evaluation and partial policy improvement) to the model-free control problem. It uses TD(0) for policy evaluation, but for learning the action-value function instead of the state-value function. The following update rule characterizes how TD(0) can be adapted for learning the action-value function:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)). \tag{1.8}$$

To use this update rule after executing action a_t in state s_t, we need to observe R_t, the next state s_{t+1}, and decide on the next action a_{t+1} according to the current policy. While interacting with the environment, the SARSA[1] algorithm applies the update rule (1.8) for every executed action, and then updates the policy to be ϵ-greedy with respect to the new estimate of the action-value function. We then execute the action that was selected by the old policy and apply new updates to the action-value function and the policy, and so on till convergence. Aside from the standard conditions (1.1) on the sequence of step-size parameters, SARSA can be shown to converge to an optimal policy as long as all state-action pairs are eventually visited an infinite number of times and the policy converges in the limit to a greedy policy, which can be achieved by setting ϵ as $\frac{1}{t}$.

[1]The acronym SARSA comes from the quantities used in update rule (1.8): the current state, the chosen action, the collected return, the next state, and the next chosen action.

1.4.2 Q-learning

In the off-policy setting, the policy we are using to interact with the environment (called the behavior policy) is different from the one we are attempting to evaluate or improve (called the target policy). This can allow learning from past experiences or by observing experts or other agents. Off-policy methods do not need to optimize ϵ-soft policies like on-policy methods; we can behave using an exploratory policy but the target policy can become an optimal one. The main challenge, however, is that the data collected is from a different policy than the target one. For SARSA, the action a_{t+1} in update rule (1.8) is chosen by the behavior policy not the target one, thus we need a way to correct for this distribution mismatch. The common way to deal with this mismatch is the use of importance sampling. The *Q-learning* algorithm is an off-policy algorithm that does not require the use importance sampling. Its update rule is similar to SARSA's except that it does not use a_{t+1} in the TD(0) target, instead it uses the optimal (greedy) action to execute in state s_{t+1} according to our current estimate of the action-value function Q. Its update rule is then:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)).$$

The behavior policy is usually taken to be the ϵ-greedy one with respect to the current estimate of Q, while the target policy is implicitly taken as the greedy one. Q-learning has been shown to converge with probability 1 to the optimal action-value function assuming that all state-action pairs continue to be updated, along with a variant of the usual conditions on the sequence of step-size parameters.

The way that Q-learning directly approximates the optimal action-value function bears a lot of resemblance to the value iteration algorithm that was discussed before. However, by looking at the Q-learning update rule, one can see the various adaptations devised to deal with the challenges that we have faced when moving to the model-free setting compared to the setting where dynamic programming methods are applicable. One adaptation is approximating the action-value function instead of the state-value function, so that we can obtain the greedy policy without knowledge of the model. Another adaptation concerns the use of only one sample of the next state (s_{t+1}) instead of a weighted average over the possible next states like in the Bellman optimality operator. Also, we update state-action pairs asynchronously as soon as they are visited by the behavior policy, which we use to ensure sufficient exploration. And lastly, we perform soft updates controlled by a step-size parameter to ensure convergence.

Chapter 2

Reinforcement Learning II

The policy evaluation methods that were considered so far rely on a tabular representation of the state space, and thus cease to be practical for large state spaces. One difficulty is the huge amount of storage required to keep track of the value of each state. Another difficulty is the time and resources required to approximate the value of each state accurately enough, which would require us, in simulation-based methods, to encounter each state (in an enormous state space) a sufficient number of times. In a lot of interesting applications, the state space is infinite; either a countable set or a more general case like a subset of a Euclidean space. In such cases, of course, tabular methods are not applicable. Moreover, when the action space is large or infinite, the policy improvement methods that we have discussed are, in general, not practical anymore. A third problem concerns the objective that we have adopted so far. In a lot of practical applications, focusing solely on achieving high expected return is not adequate without safety guarantees.

In this chapter, we go beyond the basic methods of the previous chapter and attempt to address the three problems we have just highlighted. In particular, we start by describing policy evaluation methods that use function approximation, which makes them applicable to large or even infinite state spaces. We then discuss policy gradient methods, which provide a different approach for the control problem that is suitable when the action space is large. Moreover, these methods enjoy more robust convergence properties in the function approximation setting compared to the value-based policy improvement methods of the last chapter. Lastly, we shift our focus to the problem of risk-aversion in reinforcement learning. We justify the need for adopting alternative objectives in practical applications, and provide a brief overview of the relevant methods in the literature.

2.1 Function Approximation Methods for Policy Evaluation

Before describing specific policy evaluation algorithms, we start with a brief discussion on the function approximation paradigm. In general, what distinguishes function approximation methods from tabular ones is their ability to make general predictions from limited knowledge. To this end, it is crucial to build a compact and mathematically convenient representation of the state and/or action spaces. This is usually done via a number of functions describing various *features* of the elements of the space. We would generally expect it to be a good idea to make similar predictions for two elements with similar (in some sense) features. A good feature representation thus conveys about an element the relevant information for the problem at hand. For example, in the policy evaluation problem, the feature representation of a state should describe aspects of a state that are relevant for predicting the expected return from that state. Given such a representation, we use function approximation methods to learn and approximate some unknown function of the elements. This can be the action-value function for state-action pairs, or a function mapping states to desired probability distributions over actions (i.e. a desired policy).

The function approximation setting can generally be characterized by three major components. The first being a function space (possessing convenient mathematical properties) in which we restrict our search. The second is a performance measure (or a loss function) that guides the search (or optimization) process and provides a way to order and compare candidate functions. Generally, we do no expect our class of functions to contain the target function, and thus our goal reduces to finding the candidate within our function space with the smallest loss (or highest performance). Since, in general, we do have to settle for sub-optimal solutions, we can use the performance measure to indicate what aspects of the performance are more important for us. For example, we do not expect to obtain the true value function, but we can emphasize through the loss function that we care more about reducing the prediction error at states that are visited most often by the policy. As another example, when searching for an optimal policy, we can express (through the performance measure) that we care more about the performance of the policy at states that we are more likely to start an episode from. The third component is the data we use to obtain information about the performance of the candidate functions. They can be in the form of input-output examples of the target function, but in reinforcement learning, they are usually less direct clues. We would then use an adequate optimization procedure to automatically search for the best (according to our performance measure) function in our function space using the data that is available or being collected during the learning process.

This framework provides us freedom to manage the trade-off between the complexity of the methods (depending on the richness of the features representation and the capacity of the space of candidate functions) and the approximation error (that describes the best that we can do). What has been described so far is akin to the usual problem setting in the supervised learning field. However, in reinforcement learning, we face a unique set of challenges. Usually, we are not readily provided with a dataset, instead, the sampling process is interleaved with and can be influenced by the learning process. Also, as mentioned before, we often do not have readily available targets to train the predictor, we may instead rely on bootstrapping. A related challenge is that of non-stationarity that may arise when using bootstrapping or when the target function changes, like in policy iteration methods. Another issue is the temporal dependence between the collected data since they are usually obtained by interacting with the MDP in a sequential manner.

In the following, we focus on function approximation methods for learning the state-value function for a given stationary policy π. These methods are applicable for large or infinite spaces. However, in this section we assume the state space is finite just so we can adopt the more convenient matrix notation, specially that most of the relevant results in the literature and presented in that format. In any case, extending the algorithms and their results to more general state spaces should be straightforward, possibly requiring some additional technical assumptions. In general, we consider functions of the form $V(s,\omega)$ where s is a state and $\omega \in \mathbb{R}^d$ is a parameter vector, and denote by V_ω the function over the states determined by the parameter vector ω (i.e. $V_\omega(\cdot) = V(\cdot,\omega)$). This ω could be the weights of a neural network that takes the state as an input and outputs its value, or it could be the weight vector of a linear function outputting the value of a state as the dot product of its feature vector with the weight vector. In any case, and as discussed before, we need to define a loss function with which we can evaluate the candidate functions. Suppose that our Markov chain (the MDP when acting with policy π) has a stationary distribution $\mu(\cdot)$ over the states, we can define the following *mean square value error* [1]:

$$\overline{\mathrm{VE}}(\omega) = \sum_{s\in\mathcal{S}} \mu(s)[V^\pi(s) - V(s,\omega)]^2.$$

The ideal goal will be to find a global minimum of $\overline{\mathrm{VE}}$, i.e. some ω^* such that $\overline{\mathrm{VE}}(\omega^*) \leq \overline{\mathrm{VE}}(\omega)\ \forall\omega \in \mathbb{R}^d$. However, in a lot of cases, finding a global minimum is generally not possible, and we have to settle for a local minimum, i.e. some ω^* such that $\overline{\mathrm{VE}}(\omega^*) \leq \overline{\mathrm{VE}}(\omega)$ for every ω in some neighbourhood of ω^*.

2.1.1 Stochastic Gradient Descent with Monte-Carlo Targets

Gradient descent is a simple method for finding a local minimum of a general differentiable function. It is an iterative method producing a sequence of points $\{\omega_t\}$ where each point is obtained from the previous by moving along the direction of steepest descent according to a step-size parameter. The direction of deepest descent of a function at some point is the negative of the gradient of the function at that point. Applying this to our case, an iteration of gradient descent can be written as:

$$\omega_{t+1} := \omega_t - \alpha_t \nabla\overline{\mathrm{VE}}(\omega_t),$$

where $\{\alpha_t\}$ is a sequence of step-size parameters. Stochastic gradient descent (SGD) is a variant of this algorithm that, at each iteration, uses an unbiased estimate of the gradient instead of fully computing it. In our case, the expression of $\nabla\overline{\text{VE}}(\omega)$ is given by:

$$\nabla\overline{\text{VE}}(\omega) = -2\sum_{s\in\mathcal{S}}\mu(s)[V^{\pi}(s) - V(s,\omega)]\nabla V(s,\omega),$$

which would require processing all the states. Suppose instead that the Markov chain is at steady state, then we can use $-2[V^{\pi}(s_t) - V(s_t,\omega)]\nabla V(s_t,\omega)$ as an estimate of the gradient, where s_t is the state encountered at some time t. This gives rise to the following update rule:

$$\omega_{t+1} := \omega_t + \alpha_t[V^{\pi}(s_t) - V(s_t,\omega_t)]\nabla V(s_t,\omega_t),$$

where the factor of 2 was subsumed into the learning rate. In our setting, we do not have access to the true value function, but we can find unbiased estimates of it. As we did in the tabular case, we can use the return G_t as an unbiased estimate of $V^{\pi}(s_t)$. Using such an estimate in the gradient expression, we obtain the following SGD algorithm:

$$\omega_{t+1} := \omega_t + \alpha_t[G_t - V(s_t,\omega_t)]\nabla V(s_t,\omega_t),$$

which converges to a local minimum of $\overline{\text{VE}}$ under the standard conditions (1.1) on the sequence of step-size parameters [1].

2.1.2 Semi-Gradient TD(0)

The natural next step, analogous to what was done in the tabular setting, is to use bootstrapping instead of using the full return G_t. Inspired by the TD(0) algorithm, we can define the following algorithm:

$$\omega_{t+1} := \omega_t + \alpha_t[R_t + \gamma V(s_{t+1},\omega_t) - V(s_t,\omega_t)]\nabla V(s_t,\omega_t). \tag{2.1}$$

One immediate observation is that this algorithm is not a true gradient descent algorithm since $(R_t + \gamma V(s_{t+1},\omega_t))$ is treated as a fixed target and its dependence on ω_t is ignored, thus the name semi-gradient TD(0). The use of bootstrapping is motivated, as before, by the benefits it brings like increased efficiency, applicability to continuing problems, and the ability to learn on-line. However, we have to wonder whether using bootstrapping together with function approximation is still a sound approach. In other words, does semi-gradient TD(0) converge? And if it does, what are the characteristics of the point to which it converges? These questions can be answered for case where $V(\cdot,\omega)$ is a linear function of ω. In the following, the main insights concerning this issue are presented for the linear function approximation case mainly following the developments in [9][1].

We first need to introduce some preliminary concepts. Consider the basis functions $\varphi_i : \mathcal{S} \to \mathbb{R}, i = 1,...,d$ defined over the states, and define $\phi(.) := (\varphi_1(.),...,\varphi_d(.))^{\intercal}$ as the corresponding feature mapping. Furthermore, we assume that the basis functions are all linearly independent. Keeping in mind the assumption that the state space is

[1]In [9], they analyzed the more general algorithm of TD(λ). The relevant results for this work are simply obtained by setting λ as 0.

finite, define Φ as a $|\mathcal{S}| \times d$ matrix where each row is the feature vector of a state. Note that we can think of any function over the states as a vector of length $|\mathcal{S}|$. In the linear function approximation case, we consider functions of the form $V(.,\omega) = \phi(.)^\intercal\omega$. In other words, we restrict our search to the space $\Gamma := \{\Phi\omega : \omega \in \mathbb{R}^d\}$. Consider the $|\mathcal{S}| \times |\mathcal{S}|$ diagonal matrix D with diagonal entries $\mu(.)$ for each state, and define the inner product $\langle x, y\rangle_D := x^\intercal D y \ \forall x, y \in \mathbb{R}^{|\mathcal{S}|}$ and its associated norm $\|.\|_D := \sqrt{\langle ., .\rangle_D}$. We can then define the projection matrix $\Pi := \Phi(\Phi^\intercal D\Phi)^{-1}\Phi^\intercal D$ such that, given some $V \in \mathbb{R}^{|\mathcal{S}|}$, ΠV is the orthogonal projection of V (with respect to the inner product $\langle x, y\rangle_D$) to Γ. Note that $\langle V', V - \Pi V\rangle_D = 0 \ \forall V' \in \Gamma$ (i.e. ΠV is *D-orthogonal* to Γ), and $\Pi V = \arg\min_{V'\in\Gamma} \|V - V'\|_D$. Hence, $\|\Pi V^\pi - V^\pi\|_D^2 = min_{V'\in\Gamma}\|V' - V^\pi\|_D^2 = min_\omega \overline{\text{VE}}(\omega)$, which means that ΠV^π is the best approximation of V^π in Γ with respect to the loss function $\overline{\text{VE}}$.

Returning back to our problem, we can rewrite (2.1) for the linear case as:

$$\omega_{t+1} := \omega_t + \alpha_t[R_t + \gamma\phi(s_{t+1})^\intercal\omega_t - \phi(s_t)^\intercal\omega_t]\phi(s_t). \tag{2.2}$$

Define X_t as a shorthand notation for the tuple (s_t, r_t, s_{t+1}), and then define $g(\omega, X_t) := [R_t + \gamma\phi(s_{t+1})^\intercal\omega - \phi(s_t)^\intercal\omega]\phi(s_t)$. We can thus rewrite (2.2) as $\omega_{t+1} := \omega_t + \alpha_t g(\omega_t, X_t)$. The starting point to understanding the dynamics of the algorithm is to inspect its expected behaviour when the system is in steady-state. That is, considering, for some fixed ω, the quantity $\mathbb{E}[g(\omega, X_t)]$ when the samples are generated according to the stationary distribution of the Markov chain. Accordingly, we can study a deterministic version of the algorithm:

$$\omega_{t+1} := \omega_t + \alpha_t\, \mathbb{E}[g(\omega_t, X_t)]. \tag{2.3}$$

Under some technical assumptions, the convergence of (2.3) implies the convergence of (2.2) [9]. Note that we can write $\mathbb{E}[g(\omega, X_t)]$ in the following form:

$$\mathbb{E}[g(\omega, X_t)] = A\omega + b, \tag{2.4}$$

where $A := \mathbb{E}[\phi(s_t)(\gamma\phi(s_{t+1})^\intercal - \phi(s_t)^\intercal)]$ and $b := \mathbb{E}[\phi(s_t)R_t]$. This form does not give much intuitive insight, but under appropriate conditions, the algorithm can be show to converge if the matrix A is negative definite [9]. A more insightful relation is that

$$\mathbb{E}[g(\omega, X_t)] = \Phi^\intercal D(T^\pi(\Phi\omega) - \Phi\omega), \tag{2.5}$$

where T^π is the Bellman operator (defined in (1.5)) introduced in the previous chapter. We can use this to interpret (2.3) as a gradient descent iteration that aims at minimizing the loss function $\frac{1}{2}\|T^\pi(\Phi\omega_t) - \Phi\omega\|_D^2$, where ω_t is fixed and ω is the variable. This can be intuitively justified if we think of $T^\pi(\Phi\omega_t)$ as a better estimate of V^π than our current estimate $\Phi\omega_t$ (remember from proposition (1.1-ii) that T^π is a contraction in the supremum norm with V^π as its fixed point). The algorithm in (2.3) thus takes our estimate one (small) step closer to that improved target (while still staying in Γ). However, after every step, ω changes and we pursue a new target. Instead, we can devise a more direct variant of the algorithm that, at each step, takes our estimate as close as it can get to the target at that step (i.e. the orthogonal projection of the target to Γ). With $\Phi\omega_t$ written as V_t, we can describe such an algorithm as follows:

$$V_{t+1} = \Pi T^{\pi}(V_t), \tag{2.6}$$

where, again, its convergence properties are related to those of (2.3) [9]. This stripped down version of the algorithm exposes the central contributor to the dynamics (and convergence properties) of the algorithm: the ΠT^{π} operator. It is, in fact, not difficult to show that such an operator is a contraction in the $\|.\|_D$ norm. Specializing Lemma 4 in [9] for the case of $\lambda = 0$, we get that for any $V, V' \in \mathbb{R}^{|\mathcal{S}|}$,

$$\|T^{\pi}V - T^{\pi}V'\|_D \leq \gamma\|V - V'\|_D,$$

which means that T^{π} is a contraction in the $\|.\|_D$ norm. As for Π, since $\langle \Pi V, V - \Pi V\rangle_D = 0$ for any $V \in \mathbb{R}^{|\mathcal{S}|}$, then by the Pythagorean theorem,

$$\|V\|_D^2 = \|\Pi V\|_D^2 + \|V - \Pi V\|_D^2.$$

If instead of V, we consider $V - V'$ for any $V, V' \in \mathbb{R}^{|\mathcal{S}|}$, then

$$\|\Pi V - \Pi V'\|_D^2 \leq \|V - V'\|_D^2,$$

which means that Π is non-expansive in the $\|.\|_D$ norm. Consequently, the operator ΠT^{π} is a contraction in the $\|.\|_D$ norm, and thus, it has a unique fixed point that takes the form $\Phi\omega^*$, where such ω^* is unique since the basis functions are assumed to be linearly independent.

The contraction property of ΠT^{π}, along with the equivalence of (2.4) and (2.5), is used in [9] to show that the matrix A is indeed negative definite. This is then used, along with the standard conditions (1.1) on the step-size parameters and other technical assumptions, to show that semi-gradient TD(0) converges with probability 1 to the parameter vector ω^* that uniquely satisfies $A\omega^* + b = 0$ and is also the unique fixed point of ΠT^{π}. In [1], such point is named ω_{TD}. The function defined by this parameter vector has the property that if we apply the Bellman operator to it (which can generally takes us outside Γ) and then project it back to Γ we recover the same function. This means that $T^{\pi}V_{\omega_{\mathrm{TD}}} - V_{\omega_{\mathrm{TD}}}$ is D-orthogonal to Γ since its projection is the zero vector.

For a parameter vector ω, define the Bellman error vector $\bar{\delta}_\omega \in \mathbb{R}^{|\mathcal{S}|}$ as

$$\bar{\delta}_\omega := T^{\pi}V_\omega - V_\omega.$$

We can then conveniently tailor the following *Mean Square Projected Bellman Error* loss function [1]:

$$\overline{\mathrm{PBE}}(\omega) := \|\Pi\bar{\delta}_\omega\|_D^2.$$

It can be easily seen that ω_{TD} minimizes this loss function. Actually, it reduces it all the way to zero. However, we naturally have to wonder how meaningful it is to minimize such a function. More specifically, how does ω_{TD} perform according to our original performance metric $\overline{\mathrm{VE}}$? One intuitive insight is that since T^{π} takes $V_{\omega_{\mathrm{TD}}}$ in a direction orthogonal to Γ, then if subsequent applications of T^π (which should leads us in the limit to V^π) point in a similar direction, we would then expect that $V_{\omega_{\text{TD}}}$ should not be too far from ΠV^π [10]. The following bound provides a more formal answer [9][2]:

$$\|V_{\omega_{\text{TD}}} - V^\pi\|_D \leq \frac{1}{\sqrt{1-\gamma^2}} \|\Pi V^\pi - V^\pi\|_D.$$

In other words, $\overline{\text{VE}}(\omega_{\text{TD}}) \leq \frac{1}{1-\gamma^2} \min_\omega \overline{\text{VE}}(\omega)$. While the point that semi-gradient TD(0) converges to is generally different from the global minimum of $\overline{\text{VE}}$ (albeit at a bounded distance from it), the Monte-Carlo algorithm described in the previous section does in fact converge to the global minimum of $\overline{\text{VE}}$ (when considering linear function approximation). This, however, comes at the cost of higher variance and less efficiency.

2.1.3 Mini-Batch Sampling and the Sample Complexity of Semi-Gradient TD(0)

In this section, we describe a simple variant of the semi-gradient TD(0) algorithm that uses a mini-batch in the update instead of a single sample. Assuming the same linear function approximation setting of the previous section, we can describe the mini-batch variant (as used, for example, in [11]) with the following update rule:

$$\omega_{i+1} := \omega_i + \alpha \frac{1}{M} \sum_{t=0}^{M-1} [R_{i,t} + \gamma \phi(s_{i,t+1})^\intercal \omega_i - \phi(s_{i,t})^\intercal \omega_i] \phi(s_{i,t}), \tag{2.7}$$

where M is the mini-batch size. The only difference here is that at each iteration, we average the updates that would result from the next M interactions instead of using only a single interaction to perform the update. Intuitively, this modification should decrease the variance of the updates. In a way, it provides a middle ground between the one-sample stochastic iterates in (2.2) and the expected steady-state iterates in (2.3). To explain the justification for using it in [11], it is better to first highlight a few points concerning the finite sample complexity of semi-gradient TD(0).

In the literature, there has been a relatively recent surge of interest in the finite sample analysis of temporal difference methods with linear function approximation. A particularly nice exposition of the subject is provided in [12]. There, they start with analyzing the deterministic variant (2.3) aiming to bound the quantity $\|\omega^* - \omega_T\|_2^2$ where ω^* is the TD fixed point (the one described in the previous section) and ω_T is the estimate obtained after T iterations of the algorithm. With $\bar{g}(\omega_t) := \mathbb{E}[g(\omega_t, X_t)]$, a starting point for obtaining such a bound is the following decomposition:

$$\begin{aligned}\|\omega^* - \omega_{t+1}\|_2^2 &= \|\omega^* - \omega_t - \alpha \bar{g}(\omega_t)\|_2^2 \\ &= \|\omega^* - \omega_t\|_2^2 - 2\alpha(\omega^* - \omega_t)^\intercal \bar{g}(\omega_t) + \alpha^2 \|\bar{g}(\omega_t)\|_2^2.\end{aligned}$$

They proceed by bounding the second and the third terms above in terms of $\|\omega^* - \omega_t\|_2^2$. The resulting recursion allows for a simple bound implying a geometric convergence

[2]This bound was reported in a footnote in [9] as a tighter version of the bound in lemma 6 of the same paper.

rate. When moving to the stochastic version (aiming to bound $\mathbb{E}\big[\|\omega^* - \omega_T\|_2^2\big]$), two extra sources of error emerge: the variance of the updates and the bias resulting from the temporal dependence between the samples used across the iterations. The added error cannot be reduced unless the step-size parameters decay over time, which affects the convergence rate.

In [11], although their analysis proceeds in a slightly different manner, the use of mini-batch updates provides a way to reduce the effects of the aforementioned sources of error by increasing the batch size M. In this way, the iterates can get arbitrarily close, in expectation, to the fixed point by increasing M while using a constant step-size chosen independently of the accuracy requirement. Their analysis results in a sample complexity that is $\mathcal{O}\big(\epsilon^{-1}\log\big(\epsilon^{-1}\big)\big)$ for having that $\mathbb{E}\big[\|\omega^* - \omega_T\|_2^2\big] \leq \epsilon$, where the total number of used samples is $T \times M$. They claim an improvement of $\mathcal{O}\big(\log\big(\epsilon^{-1}\big)\big)$ over the best known complexity results in the literature at the time thanks to the use of mini-batch updates. This, however, comes at the cost of having an extra parameter to tune (namely M), and a slight betrayal of the on-line spirit of TD(0). Later on, we will extend some of the work in [11] to a risk-averse setting.

2.1.4 Least-Squares Temporal Difference

Since we know a great deal about the temporal difference fixed point when using linear function approximation, we can attempt to directly estimate it from data. This idea is adopted by the Least-Squares Temporal Difference (LSTD) algorithm [13][3]. We know from the previous section that

$$A\omega_{\text{TD}} + b = 0.$$

We can thus build the following estimates from all the samples encountered up to time t:

$$\hat{A}_t := \sum_{k=0}^{t-1} \phi(s_k)(\gamma\phi(s_{k+1})^\intercal - \phi(s_k)^\intercal)$$
$$\hat{b}_t := \sum_{k=0}^{t-1} \phi(s_k)R_k.$$

We can then compute the following estimate of ω_{TD}:

$$\omega_t := -\hat{A}_t^{-1}\hat{b}_t, \tag{2.8}$$

where Singular Value Decomposition (SVD) can be used to robustly invert $\hat{A}_t$. Note that we do not need to divide $\hat{A}_t$ and $\hat{b}_t$ by t since the two factors would cancel out in (2.8).

The main advantage of this algorithm is that it makes efficient use of samples [14]; it extracts more information from experience compared to semi-gradient methods. Another advantage is that LSTD does not use a step-size parameter, which can require tedious

[3]LSTD was originally derived in [14] in a different manner.

tuning in some problems. Its main drawback, however, is its computational complexity: building the estimates has complexity that is $\mathcal{O}(d^2)$ (where d is the dimensionality of the weight vector) and the matrix inversion is generally of $\mathcal{O}(d^3)$ complexity. It is possible to build the inverse of $\hat{A}_t$ in an incremental way, thus requiring only $\mathcal{O}(d^2)$ computations [1]. However, this is still more expensive than the $\mathcal{O}(d)$ per-step complexity of semi-gradient methods. Another point is that in non-stationary problems (like generalized policy iteration), LSTD weighs all the samples seen so far equally; it does not "forget" old samples. When LSTD is used in such problems, it is usually combined with some forgetting mechanism playing a role not unlike that of the step-size parameter [1].

2.1.5 Bellman Error Minimization

It is well-known that semi-gradient temporal difference can diverge when using non-linear function approximation or under off-policy training [9]. True gradient descent methods, however, have more robust convergence properties and would not suffer from the mentioned divergence issues. In this section, we briefly describe one possible true SGD method and discuss its shortcomings. The objective of this method concerns the Bellman error vector (or the Bellman residual) $\bar{\delta}_\omega$ that was defined before as $\bar{\delta}_\omega := T^\pi V_\omega - V_\omega$. In particular, we aim to minimize the *Mean Squared Bellman Error* [1]:

$$\overline{\mathrm{BE}}(\omega) := \|\bar{\delta}_\omega\|_D^2.$$

But, again, why is minimizing this loss function a good idea? The first reassuring thing is that, at the very least, the true value function V^π achieves zero loss since it is the fixed point of T^π. More importantly, we can motivate the pursuit of minimizing the norm of the bellman residual by the following bound [15]:

$$\|V - V^\pi\|_\infty \leq \frac{\|V - T^\pi V\|_\infty}{1-\gamma},$$

which asserts that the distance, in the supremum norm, between V (which is some arbitrary value function) and V^π can be bounded using the supremum norm of the bellman residual of V.

An SGD iteration aiming to minimize $\overline{\mathrm{BE}}(\omega)$ can be written as:

$$\omega_{t+1} := \omega_t + \alpha_t[\mathbb{E}[R_t + \gamma V(s_{t+1}, \omega_t)] - V(s_t, \omega_t)][\nabla V(s_t, \omega_t) - \gamma\, \mathbb{E}[\nabla V(s_{t+1}, \omega_t)]].$$

Since we do not have access to the environment's model, we have to use samples as estimates to the expectations in the update rule. However, to get unbiased updates, we must obtain two samples of s_{t+1}, which is unpractical in a lot of situations where we can only interact with the environment in a sequential manner and do not have the ability to reset the environment to any state of our choosing. If we only use a single sample for s_{t+1}, then, in effect, the algorithm will pursue minimizing a different loss function (named $\overline{\mathrm{TDE}}$ in [1]) at which the true value function, in general, does not achieve zero loss and is not necessarily its minimizer (even if the true value function is in the space of candidate functions). Another issue with Bellman residual minimization algorithms is that they can be quite slow [16].

To conclude our discussion of the prediction problem, we can present the diagram in figure (2.1), due to [1], as a summary of the main concepts we covered for the case of linear function approximation (with a very slight difference in notation). In this example, they consider an MDP of 3 states such that the 3D space in the figure contains all possible value functions. They then consider a linear function approximation scheme with a two dimensional weight vector ω. Hence, the 2D plane in the figure represents all the value functions that can be represented as V_ω. Starting from any value function in the plane and applying repeatedly the Bellman operator T^π (Named B^π in the figure) will lead us in the limit to the true value function V^π. Projecting V^π to the plane lands us at the point ΠV^π which minimizes $\overline{\text{VE}}$ in the plane. (Note that Monte-Carlo methods converge to that point.) Alternatively, if we do a projection step after each application of the Bellman operator, we converge to the TD fixed ω_{TD} at which $\overline{\text{PBE}}$ is zero. Represented also in the figure are the minimizers of $\overline{\text{BE}}$ and $\overline{\text{TDE}}$ in the plane.

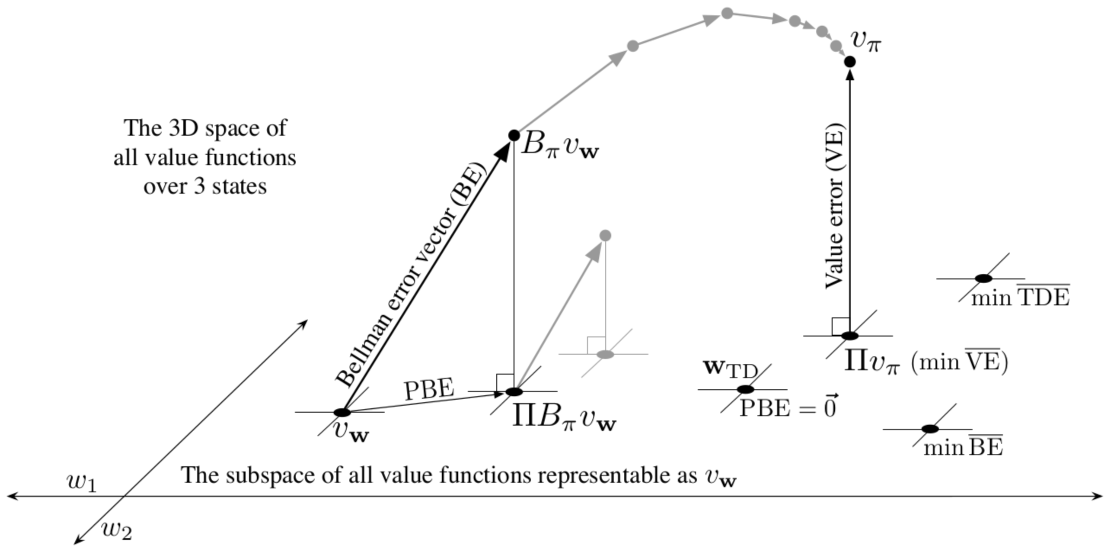
v_π
The 3D space of all value functions over 3 states
$B_\pi v_{\mathbf{w}}$
Bellman error vector (BE)
Value error (VE)
$\min \overline{\text{TDE}}$
Πv_π $(\min \overline{\text{VE}})$
$\mathbf{w}_{\text{TD}}$
PBE
$\overline{\text{PBE}} = \vec{0}$
$v_{\mathbf{w}}$
$\Pi B_\pi v_{\mathbf{w}}$
$\min \overline{\text{BE}}$
w_1
The subspace of all value functions representable as $v_{\mathbf{w}}$
w_2

Figure 2.1: The geometry of linear value function approximation [1].

2.2 Policy Gradient Methods

We now shift our focus to the problem of approximating optimal policies. In principle, the value function approximation schemes discussed in the previous section can be utilized within the generalized policy iteration framework in various ways. Examples of such methods in the literature include the fitted Q-iteration algorithm [17] and the least squares policy iteration algorithm [10]. These methods, however, usually cease to be practical when the action space is large or continuous since most of these methods, one way or another, rely on searching for the best action to take in a state according to some estimate of the value function. Alternatively, we focus in this section on the policy search framework, which can naturally handle large or continuous action spaces. Differently from all the methods we considered so far (which are referred to as value-based methods), policy search methods look for the optimal policy, according to some performance measure, directly within the space of policies without necessarily relying on an estimate of the value function, although they usually benefit from having such an estimate [1].

In policy search methods, we restrict our search to a chosen class of parameterized policies Π, such that $\pi(.|s,\theta)$ (or $\pi_\theta(.|s)$) denotes the probability mass (or density) function of the action to be taken in state s, where $\theta \in \mathbb{R}^{d_\theta}$ is a parameter vector characterizing the policy. The chosen class of policies usually possesses convenient mathematical properties, like $\pi(.|s,\theta)$ being continuously differentiable with respect to θ for any state s and action a. For example, if we have a one-dimensional continuous action space, we may consider $\pi(.|s,\theta)$ as the probability density function of a Gaussian distribution whose parameters depend on the state s and the parameter vector θ. In other words,

$$\pi(a|s,\theta) = \frac{1}{\sigma(s,\theta)\sqrt{2\pi}} \exp\left(-\frac{(a-\mu(s,\theta))^2}{2\sigma(s,\theta)^2}\right),$$

where σ and μ are parameterized functions that define the standard deviation and the mean of the distribution. Such functions can, for example, take the form of a neural network having θ (or a portion of it) as its weights and the feature mapping of a state as input.

To be more specific, we focus on policy gradient methods. Given a performance measure $J(\theta)$, these methods attempt to find the optimal policy within Π by doing gradient *ascent* on $J(\theta)$. In other words, they update the policy parameters as follows:

$$\theta_{t+1} := \theta_t + \alpha_t \nabla_\theta J(\theta_t), \tag{2.9}$$

where as usual, we can replace $\nabla_\theta J(\theta_t)$ by some unbiased estimate of it, thus performing *stochastic* gradient ascent. Aside from being able to naturally handle large action spaces, another advantage of policy gradient methods is that the action probabilities change smoothly when updating the parameters compared to value-based methods where the action probabilities can change abruptly with small change in the value function due to the greedy action selection process. This allows policy gradient methods to have stronger convergence guarantees [1]. One disadvantage of policy gradient methods is that performing gradient ascent can get us stuck in poor local optima. However, it can be argued that this problem can, in principle, be avoided by increasing exploration and the representational power of the policy parameterization [18].

2.2.1 The Policy Gradient Theorem

In order to be able to use (2.9), we need to find an expression for the gradient of the performance measure with respect to the policy parameters. In continuing discounted problems, the performance measure is usually defined as

$$J(\theta) := \mathop{\mathbb{E}}_{\substack{s_0 \sim \mu_0 \\ a_t \sim \pi_\theta(\cdot|s_t) \\ s_{t+1} \sim P(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right], \tag{2.10}$$

where μ_0 is the initial state distribution. The gradient of $J(\theta)$ is given by the policy gradient theorem [19]:

$$\nabla_\theta J(\theta) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s \sim d_{\mu_0, \pi_\theta}(\cdot) \\ a \sim \pi_\theta(\cdot|s)}} [Q^{\pi_\theta}(s,a) \nabla_\theta \log \pi_\theta(a|s)], \tag{2.11}$$

where d_{μ_0,π_θ} is the discounted state distribution defined as:

$$d_{\mu_0,\pi_\theta}(s) := (1-\gamma) \mathop{\mathbb{E}}_{s_0 \sim \mu_0(\cdot)} \left[\sum_{t=0}^{\infty} \gamma^t p_{\pi_\theta}(s_0 \xrightarrow{t} s) \right], \tag{2.12}$$

where $p_\pi(s_0 \xrightarrow{t} s)$ is the probability of ending up in s after t steps starting from s_0 and executing policy π_θ. Note that in (2.11), we do not have to compute the gradient of any quantity that depends on the environment's model. Also note that the form of the policy gradient theorem is convenient for obtaining estimates of the gradient via sampling. However, the states we encounter need to be sampled form the discounted state distribution, which is not the distribution of states that results from interacting naturally with the MDP using the policy. One way of forcing the encountered states to obey the discounted state distribution is to interact normally with the MDP, but at each step and with probability $1-\gamma$, we truncate the episode and draw again from the initial state distribution [18]. That is, we interact with a modified MDP having the following transition kernel:

$$\tilde{P}(\cdot|s,a) = \gamma P(\cdot|s,a) + (1-\gamma)\mu_0(\cdot),$$

where P is the transition kernel of the original MDP. A slightly less wasteful way is to sample normally from the on-policy distribution but discount the gradient estimates according to the time-step in which the sampled state-action pair was encountered (see (2.13) for an example). An argument for this can be found in [18] for a similar problem.

2.2.2 REINFORCE

The only missing piece now is the true action value function Q^{π_θ}, which we do not have access to. As usual, the simplest way to deal with this is to use unbiased Monte-Carlo estimates of the action-value function. This gives rise to the following algorithm:

$$\theta_{t+1} := \theta_t + \alpha_t \gamma^t G_t \nabla_\theta \log \pi_{\theta_t}(a_t|s_t), \tag{2.13}$$

where G_t is the return starting from time t. This is one form of the REINFORCE algorithm [20]. Convergence of such an algorithm to a local optimum is assured under standard conditions on the reduction of the step-size parameters [1]. Being a Monte-Carlo method, this algorithm is not directly applicable to continuing problems and suffers from high variance. The latter problem can be partially mitigated by using baselines. For any state-dependent baseline $b : \mathcal{S} \to \mathbb{R}$, it can be shown that

$$\mathop{\mathbb{E}}_{a \sim \pi_\theta(\cdot|s)} [b(s) \nabla_\theta \log \pi_\theta(a|s)] = \mathbf{0}. \tag{2.14}$$

Knowing this, we can rewrite the gradient as follows:

$$\nabla_\theta J(\theta) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s \sim d_{\mu_0, \pi_\theta}(\cdot) \\ a \sim \pi_\theta(\cdot|s)}} [(Q^{\pi_\theta}(s,a) - b(s)) \nabla_\theta \log \pi_\theta(a|s)], \tag{2.15}$$

which means that subtracting a baseline from the action-value function does not bias the gradient as long as the baseline does not depend on the actions. However, the choice of the baseline can have an effect on the variance of the algorithm. In practice, a natural choice for the baseline is an estimate for the state-value function [1]. This results in the following algorithm:

$$\theta_{t+1} := \theta_t + \alpha_t \gamma^t (G_t - V_{\omega_t}(s_t)) \nabla_\theta \log \pi_{\theta_t}(a_t|s_t), \tag{2.16}$$

where V_{ω_t} is an estimate (with current parameter vector ω_t) of the state-value function of π_{θ_t} learned using any of the methods we described for policy evaluation with function approximation. Note that the term $G_t - V_{\omega_t}(s_t)$ can be seen as an estimate of the advantage function at (s_t, a_t), where the advantage function is defined as:

$$A^\pi(s,a) := Q^\pi(s,a) - V^\pi(s).$$

Thus, unlike in (2.13) where we move the parameters along the direction [4] of $\nabla_\theta \log \pi_\theta(a|s)$ proportional to the value of the action, we now move proportional to the difference between the value of the action and the value of the state. The algorithm in (2.16) has the same convergence guarantees of the plain REINFORCE algorithm, but it can learn a lot faster in practice due to its lower variance.

2.2.3 Actor-Critic Methods

The natural next step is to replace the Monte-Carlo estimates of the action value function with some bootstrapped target. Since

$$Q^{\pi_\theta}(s,a) = R(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim P(\cdot|s,a)} \left[V^{\pi_\theta}(s') \right],$$

we can use $R(s_t, a_t) + \gamma V^{\pi_\theta}(s_{t+1})$ as an unbiased estimate of $Q^{\pi_\theta}(s_t, a_t)$. But since we do not have access to V^{π_θ}, we can substitute it with our estimate V_ω that we are already using as a baseline. The resulting algorithm takes the following form:

$$\theta_{t+1} := \theta_t + \alpha_t \gamma^t (R_t + \gamma V_{\omega_t}(s_{t+1}) - V_{\omega_t}(s_t)) \nabla_\theta \log \pi_{\theta_t}(a_t|s_t). \tag{2.17}$$

[4]Note that $\nabla_\theta \log \pi_\theta(a|s) = \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}$, where $\nabla_\theta \pi_\theta(a|s)$ (referred to as the *score function*) is the direction along which moving the policy parameters leads to the fastest increase in the probability of taking action a at state s, and dividing by $\pi_\theta(a|s)$ has the effect of not giving an advantage to actions that are selected more often by π_θ.

At each step, we can wait until we have an accurate enough (in some sense) estimate of the value function of the current policy before updating the policy using (2.17). Alternatively, we can update ω once at each step (for example using semi-gradient TD(0)) resulting in a two-timescale algorithm, where the policy should be updated slower than our value function estimate to allow it to "track" the changing policy.

The algorithm in (2.17) is an instance of a general class of policy gradient methods known as *actor-critic* methods. The *critic* in these methods is the value-function estimate as it is used to evaluate (or criticize) the performance of the *actor*, which is the current policy. Although these methods introduce bias due to the inaccuracy of the learned value function, they typically have less variance than Monte-Carlo methods, and they allow for on-line learning. Actor-critic methods can be considered, in some sense, as the intersection of policy-based and value-based methods. Methods like (2.17) are also referred to as *advantage actor-critic* (A2C) methods. This is because the term $R_t + \gamma V_{\omega_t}(s_{t+1}) - V_{\omega_t}(s_t)$ can be seen as an estimate of the advantage function at (s_t, a_t).

2.2.4 Compatible Function Approximation

The notion of compatible function approximation provides one way of using a critic learned using function approximation without biasing the gradient estimates. Suppose we want to learn an approximation for the advantage function $A^{\pi_\theta}(s,a)$ using a function $f_\omega(s,a)$ characterized by a parameter vector ω. It can be easily seen, as demonstrated in [19], that if

$$\nabla_\omega f_\omega(s,a) = \nabla_\theta \log \pi_\theta(a|s),$$

and

$$\mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi_\theta}(\cdot)\\ a\sim\pi_\theta(\cdot|s)}}[(A^{\pi_\theta}(s,a) - f_\omega(s,a))\nabla_\omega f_\omega(s,a)] = \mathbf{0},$$

then

$$\nabla_\theta J(\theta) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi_\theta}(\cdot)\\ a\sim\pi_\theta(\cdot|s)}}[f_\omega(s,a)\nabla_\theta \log \pi_\theta(a|s)].$$

Which means that we can use f_ω instead of A^{π_θ} without introducing any bias. We can satisfy the stated conditions if we use a linear function $f_\omega(s,a) = \phi(s,a)^\intercal\omega$, such that the (policy-dependent) feature mapping $\phi(s,a)$ for a state-action pair is $\nabla_\theta \log \pi_\theta(a|s)$, and ω is chosen as to minimize the following loss function:

$$\mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi_\theta}(\cdot)\\ a\sim\pi_\theta(\cdot|s)}}\left[(A^{\pi_\theta}(s,a) - f_\omega(s,a))^2\right].$$

We can then update the policy parameters according to

$$\theta_{t+1} := \theta_t + \alpha_t\gamma^t\nabla_\theta \log \pi_{\theta_t}(a_t|s_t)\nabla_\theta \log \pi_{\theta_t}(a_t|s_t)^\intercal\omega_t.$$

The same argument also holds for learning the action-value function instead of the advantage function. In fact, the same argument holds even when subtracting any state-dependent baseline from the action-value function. However, the form suggested above for f_ω implies that $\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[f_\omega(s,a)] = 0$, which makes it most suitable for the learning the advantage function since it can be easily seen that $\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[A^{\pi_\theta}(s,a)] = 0$.

In practice, using compatible function approximation is not necessarily the best choice to make in terms of efficiency. However, the resulting unbiasedness of the estimates of the gradient allows us to inherit the convergence properties of stochastic gradient ascent in a straightforward manner [19]. Moreover, compatible function approximation plays a convenient role when used in natural policy gradient methods, which we discuss next.

2.2.5 Natural Actor-Critic

Standard gradient ascent algorithms suffer from a number of problems; they can get stuck in plateaus, and they can take over-aggressive steps on steep ridges. One attempt towards mitigating some of these problems is the use of natural gradient methods [21]. In reinforcement learning problems, these methods move the parameters along the direction of steepest ascent according to the Fisher metric [22]. This way, we follow the direction along which the performance increases the fastest when the divergence between the distributions of possible trajectories induced by the old and the new policies is kept sufficiently small [23]. This can lead to a more stable learning process compared to standard gradient methods, in which the constraint is on the Euclidean distance between the old and the new parameter vectors. For our setting, the natural gradient is given by:

$$\nabla_\theta^{\mathrm{NG}} J(\theta) := F_\theta^{-1}\nabla_\theta J(\theta),$$

where F_θ is the *Fisher information matrix* defined as [22]:

$$F_\theta = \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi_\theta}(\cdot)\\ a\sim\pi_\theta(\cdot|s)}}[\nabla_\theta \log\pi_\theta(a|s)\nabla_\theta\log\pi_\theta(a|s)^\intercal].$$

We can then update the policy according to:

$$\theta_{t+1} := \theta_t + \alpha_t F_{\theta_t}^{-1}\nabla_\theta J(\theta_t). \tag{2.18}$$

The angle between the standard gradient and the natural one is never larger than 90 degrees, and thus the natural policy gradient algorithm can also be guaranteed to converge to a local optimum [22]. In practice, F_θ^{-1} is estimated using samples, and we can use any of the methods discussed so far to obtain estimates of $\nabla_\theta J(\theta)$. Estimating the Fisher information matrix from samples can be expensive, but that can be avoided when using a critic obeying the compatible function approximation conditions. To see why, consider that when using compatible function approximation with a linear architecture (as proposed in the previous subsection), the gradient can be written as:

$$\begin{aligned}\nabla_\theta J(\theta) &= \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi_\theta}(\cdot)\\ a\sim\pi_\theta(\cdot|s)}}[\nabla_\theta\log\pi_\theta(a|s)\nabla_\theta\log\pi_\theta(a|s)^\intercal]\,\omega\\ &= \frac{1}{1-\gamma}F_\theta\omega.\end{aligned}$$

From that, it is easy to see that (2.18) reduces to:

$$\theta_{t+1} := \theta_t + \alpha_t \omega_t,$$

where ω_t is the parameter vector for our compatible approximator of the advantage function of π_{θ_t}. Examples of practical natural actor-critic algorithms can be found in [22] and [24].

2.3 Risk Aversion in Reinforcement Learning

The methods that we have considered so far strive to, one way or another, find a policy that acts optimally in the sense that it achieves a high expected value of the return. This notion of achieving high performance in expectation is indeed the adopted one in the vast majority of the reinforcement learning literature, in all its diversity. It is not hard to see, however, that adopting a policy that acts optimally in expectation is not suitable for a lot of application domains since the policy can still lead the agent to rare but catastrophic events. The most immediate examples of such domains are the robotics and the financial domains. When applying reinforcement learning methods in these domains, it is quite imperative to have additional guarantee on the safety of the implemented policy.

We can distinguish between two notions of safety in reinforcement learning [25]. The first one is related to the training process itself. In the cases where a simulation environment is not available, the training process needs to be carried out in a high-stakes (physical or virtual) environment where bad outcomes can have ramifications in the real world. The difficulty of this problem is that reinforcement learning methods rely on extensive exploration strategies, which undoubtedly can lead to undesirable outcomes. Some approaches attempt to tackle this problem by imposing performance guarantees on the policies obtained during training (e.g. [26] and [27]), while some approaches strive to avoid states or transitions that can lead to damage via associating a safety function to each state or transition (e.g. [28]). The second notion of safety, which is the focus in this work, deals with the inherent uncertainty of the environment itself. That is, it deals with the risk due to the stochasticity of the outcomes when interacting with an MDP. Methods that attempt to deal with this kind of risk usually do so by modifying the objective to include some notion related to risk. This is usually done by considering some statistical property (e.g. the variance) of the return (other than its expected value) to optimize. The challenge is how to extend the standard reinforcement learning tools to deal with such modified objectives. In the following, we briefly mention a few methods that incorporate risk into the objective.

2.3.1 Utility-Based Objectives

A simple way to incorporate risk is to use utility functions [29]. Roughly speaking, instead of trying to maximize the expected value of a random variable, we try to maximize the expected value of a transformation (the utility function) applied to the realizations of our random variable. The main difficulty of this approach is that it is usually hard to specify a suitable utility function for the problem at hand [29]. The most common utility function in reinforcement learning is the exponential utility applied to the return (e.g. [30]). This performance measure usually takes the following form:

$$V = \frac{1}{\beta} \log\Big(\mathbb{E}[e^{\beta R}]\Big),$$

which the agents tries to maximize. Here R is the return, and $\beta \neq 0$ is a real parameter used to express the risk sensitivity. Setting $\beta < 0$ encourages risk-averse behaviour, while setting $\beta > 0$ encourages risk-seeking behaviour. When β approaches zero, the agent becomes risk-neutral, and we recover the standard objective of maximizing the expected reward. These insights are easily observed when writing the Taylor expansion of V:

$$V = \mathbb{E}[R] + \frac{\beta}{2}\mathrm{Var}[R] + O(\beta^2),$$

which shows that the way that the exponential utility controls risk-sensitivity is related to the variance, which we consider next.

2.3.2 Variance-Related Objectives

Indeed, the variance is a very conceptually simple way to control risk. Smaller variance leads to less variability of the outcomes, and hence more predictability. However, the variance is usually not the most suitable risk measure to adopt. For once, it treats the variability below the means equally with the variability above it. But in a maximization problem, we are not usually concerned with the variability above the mean. Nonetheless, the simplicity and interpretability of the variance has granted it a lot if attention in the literature, dating back to early works like [31]. One can simply adopt the variance as an objective function that is to be minimized. A more common approach is to use it together with the standard expected return objective J in a mean-risk fashion. This can take various forms [32]:

1. Maximize J such that $\mathrm{Var}[R] \leq c$.
2. Minimize $\mathrm{Var}[R] \leq c$ such that $J \geq c$.
3. Maximize the Sharpe Ratio: $\frac{J}{\sqrt{\mathrm{Var}[R]}}$.
4. Maximize $J - c\sqrt{\mathrm{Var}[R]}$.

Whichever variant one chooses to focus on, the involvement of the variance of the return brings about some difficulties compared to the risk-neutral case. To see this, we consider the problem of policy evaluation. Remember that the value function of a state s is defined[5] as the expected return when starting from s:

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \middle| s_0 = s\right].$$

[5] Assuming the policy is fixed and omitted from the notation for clarity. Moreover, we write the reward as a deterministic function of just the state for simplicity.

Instead of the expected value, we are interested in the variance of the return starting from s, which we denote by $V'(s)$. Most reinforcement learning methods are built upon the dynamic programming paradigm, for which the simple linear form of the Bellman equations of V is quite essential. The same, unfortunately, does not hold for the variance [5]. Towards devising a practical approach for learning the variance, we can write V' as:

$$V'(s) = V^{(2)}(s) - V(s)^2, \tag{2.19}$$

where $V^{(2)}(s)$ is the second moment of the return starting from s. That is,

$$V^{(2)}(s) = \mathbb{E}\left[\left(\sum_{t=0}^{\infty}\gamma^t R(s_t)\right)^2 \middle| s_0 = s\right]. \tag{2.20}$$

While the Bellman equations for V take the following familiar form:

$$V(s) = R(s) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s)V(s'),$$

similar equations for $V^{(2)}$ have been derived in [31] (see also [5]) as:

$$V^{(2)}(s) = R(s)^2 + 2\gamma R(s)\sum_{s'\in\mathcal{S}} P(s'|s)V(s') + \gamma^2 \sum_{s'\in\mathcal{S}} P(s'|s)V^{(2)}(s'). \tag{2.21}$$

The equations for $V^{(2)}$ resemble those of V if one would use a (policy-dependent) reward transformation of the form $R(s)^2 + 2\gamma R(s)\sum_{s'\in\mathcal{S}} P(s'|s)V(s')$. The approach in [5] is to use these equations to extend the theory and methodology of temporal difference learning using linear function approximation (see Subsection (2.1.2)) for the joint estimation of V and $V^{(2)}$, which are then used to estimate V' via (2.19).

As for the control problem, it has been shown in [33] that the complexity of computing a policy that maximizes the mean return under a variance constraint is NP-hard. However, policy gradient algorithms that perform local optimization are still viable. In [32], policy gradient algorithms are derived for some variance-related risk criteria in the episodic case.

2.3.3 Other Risk Measures

As we pointed earlier, a drawback of the variance as a risk measure[6] is that also penalizes the variability above the mean, which is usually irrelevant in a maximization problem. One alternative is the *lower semi-deviation*. For a random variable Z with finite p^{th} order moments, the p^{th} order lower semi-deviation is defined as [29]:

$$\sigma_p^-[Z] := \left(\mathbb{E}\left[(\mathbb{E}[Z] - Z)_+^p\right]\right)^{1/p},$$

where, for a real function f, $f_+(x) = \max(f(x), 0)$. In other words, we are only concerned with the outcomes below the mean.

[6]A risk measure, in general, can be defined as a function that maps a random variable to the extended real line $\mathbb{R}\bigcup\{-\infty, +\infty\}$.

Another alternative is the Conditional Value at Risk (CVaR). The α-quantile of a random variable Z is defined as:

$$q_\alpha \coloneqq \{x : P(Z \leq x) = \alpha\}.$$

The α-CVaR is then defined as [34]:

$$\mathrm{CVaR}_\alpha(Z) \coloneqq \mathbb{E}[Z|Z \leq q_\alpha].$$

In other words, it is the expected value of the worst $\alpha\%$ of outcomes, which makes it suitable for accounting for rare but catastrophic outcomes. Both the CVaR and the lower semi-deviation belong to a wide class of risk measures known as *coherent risk measures* [35]. This is a well-known class of risk measures in the financial domain, which satisfies certain "rationality" properties. In [36], a policy gradient method is proposed for the entire class of coherent risk measures.

2.3.4 Per-Step Reward Approaches

The last class of approaches that we discuss, which is of most relevance to this work, is the class of approaches that are concerned with the distribution of the per-step reward instead of the full return. The reward volatility introduced in [7] is concerned with the variance of the per-step reward. We will discuss the reward volatility in detail in the next chapter, where we formulate the problem to be solved in this work. For now, we highlight that, compared to the return variance, the reward volatility admits simpler Bellman-style equations, which makes adapting risk-neutral algorithms more straightforward. Moreover, it is shown in [7] that minimizing the reward volatility can be seen as a proxy for minimizing the return variance in the sense that low reward volatility imply low return variance. And like the variance, we can define a mean volatility objective that provides a way to manage the trade-off between maximizing the expected return and minimizing the reward volatility. Lastly, we can mention [37] as an example of a related approach. They consider also the mean volatility objective, and propose a block cyclic coordinate ascent approach in which risk-neutral policy evaluation and control algorithms can be used in an off-the-shelf manner. We will also see more of this approach in the next chapter.

Chapter 3

Problem Setting: The Mean-Volatility Objective

The purpose of this chapter is to formulate the problem that is to be addressed in this work. Firstly, we discuss in more detail the reward-volatility risk measure [7] and the related mean-volatility objective. We discuss how the reward-volatility compares to other risk measures, and display some of its properties. We also discuss the issues that one would face when attempting to adapt risk-neutral reinforcement learning algorithms to optimize the mean-volatility objective. We then present a simple actor-only policy gradient algorithm for optimizing the mean-volatility to serve as a starting point. Finally, we outline two different methods for policy evaluation under the mean-volatility objective. In subsequent chapters, we carry out some theoretical analysis of these methods culminating in the finite sample analysis of a complete actor-critic algorithm.

Unless otherwise stated, we assume that our MDP is characterized by the following tuple: $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu_0 \rangle$. Here, $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces which we just assume to be measurable sets. P, R, and γ are the transition kernel, reward function, and discount factor defined analogously in Section 1.1, and μ_0 is the initial state distribution. We also assume that the reward function is uniformly bounded in absolute value, which means that there is some value $R_{\max}$ such that [1] $|R(s,a)| \leq R_{\max}$ $\forall (s,a) \in S \times A$. When relevant, we assume, like in Section 2.2, that the policies we consider are from a class Π of policies parameterized by a vector $\theta \in \mathbb{R}^{d_\theta}$, and $\pi_\theta(a|s)$ is continuously differentiable with respect to θ for any state-action pair. A bit differently from Section 2.2, we define the (risk-neutral) objective as

$$J_\pi := (1-\gamma) \underset{\substack{s_0 \sim \mu_0 \\ a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(\cdot|s_t, a_t)}}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right]. \tag{3.1}$$

This definition differs from definition (2.10) by the normalization factor $(1-\gamma)$, which

[1] Remember that the reward function is assumed, for simplicity, to be a deterministic function of state-action pairs.

ensures that J_π belongs to the interval $[-R_{\max}, R_{\max}]$. Note that we drop the dependence on θ from the notation whenever it is irrelevant or clear from the context.

3.1 Reward-Volatility

In the previous chapter, we considered the variance of the return as a simple measure of uncertainty when interacting with an MDP. Since our risk-neutral performance measure J_π is the (normalized) expected value of the return when starting from the initial state distribution, we can define the return variance as follows

$$\sigma_\pi^2 := \mathop{\mathbb{E}}_{\substack{s_0 \sim \mu_0 \\ a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(\cdot|s_t, a_t)}} \left[\left(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) - \frac{J_\pi}{1-\gamma} \right)^2 \right]. \tag{3.2}$$

If we adopt such a risk measure, we would then look for policy that achieves optimal σ_π^2 (i.e. low variance). Alternatively, as we discussed in the previous chapter, we can construct a mean-variance performance measure that takes the form $J_\pi - \lambda\sigma_\pi^2$, where $\lambda \geq 0$ is a parameter via which we can express how much we care about reducing the variance compared to increasing the expected return. We know from our discussion in the previous chapter that adapting value-based risk-neutral algorithms for optimizing either σ_π^2 or $J_\pi - \lambda\sigma_\pi^2$ is not directly possible since the variance of the return starting from a certain state lacks the simple linear Bellman equations of the risk-neutral setting. Moreover, in policy-based methods, the policy gradients for these objectives do not take the same simple form of (2.11) in the risk-neutral case [32].

In this thesis, we adopt the reward volatility introduced in [7] as our risk measure, which is concerned with the variance of the per-step reward instead of the variance of the return. Recalling the definition of the discounted state distribution $d_{\mu_0,\pi}(\cdot)$ in (2.12), we can rewrite (3.1) as:

$$J_\pi = \mathop{\mathbb{E}}_{\substack{s \sim d_{\mu_0,\pi}(\cdot) \\ a \sim \pi(\cdot|s)}} [R(s, a)]. \tag{3.3}$$

That is, J_π can be seen as the expected value of the step reward random variable where the states are drawn from the discounted state distribution (which depends on the policy) and the actions are drawn according to the policy. The reward volatility ν_π^2 is the variance of this random variable. In other words,

$$\nu_\pi^2 := \mathop{\mathbb{E}}_{\substack{s \sim d_{\mu_0,\pi}(\cdot) \\ a \sim \pi(\cdot|s)}} \left[(R(s, a) - J_\pi)^2 \right]. \tag{3.4}$$

Whereas σ_π^2 is concerned with the variance of the accumulated reward regardless of the fluctuations of the step rewards, ν_π^2 takes into account the variance of the intermediate results, which can be relevant in some applications [7]. And while the reward volatility does not convey information about the correlation between step rewards, the following bound (Lemma 1 in [7]) justifies its use as a proxy for the return variance:

$$\sigma_\pi^2 \leq \frac{\nu_\pi^2}{(1-\gamma)^2}.$$

Which means that low reward volatility implies low return variance. Note that the converse is generally not true.

Similar to J_π, we can write ν_π^2 in an alternative form:

$$\nu_\pi^2 = (1-\gamma) \underset{\substack{s_0 \sim \mu_0 \\ a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(\cdot|s_t, a_t)}}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t) - J_\pi)^2 \right].$$

Motivated by this, we can define the action-volatility function X^π as:

$$\begin{aligned} X^\pi(s,a) &:= \underset{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(\cdot|s_t, a_t)}}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t) - J_\pi)^2 \middle| s_0 = s, a_0 = a \right] \\ &= (R(s,a) - J_\pi)^2 + \gamma \underset{\substack{s' \sim P(\cdot|s,a) \\ a' \sim \pi(\cdot|s')}}{\mathbb{E}} \left[X^\pi(s', a') \right], \end{aligned}$$

where the latter form is a Bellman equation akin to that of the action-value function in the risk-neutral case. The difference here is that the rewards undergo a policy-dependent transformation $(R(s,a) - J_\pi)^2$. Similarly, we can define the state-volatility function W^π as:

$$W^\pi(s) := \underset{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim P(\cdot|s_t, a_t)}}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t) - J_\pi)^2 \middle| s_0 = s \right].$$

Similar to the mean-variance, we can define the mean-volatility performance measure:

$$\eta_\pi = J_\pi - \lambda \nu_\pi^2,$$

where $\lambda \geq 0$ is again a parameter managing the trade-off between maximizing J_π and minimizing ν_π^2. Accordingly, we can define the transformed state-value function $V_\pi^\lambda(s) := V^\pi(s) - \lambda W^\pi(s)$, and the transformed action-value function $Q_\pi^\lambda(s,a) := Q^\pi(s,a) - \lambda X^\pi(s,a)$. Note that these value functions also admit simple Bellman equations, unlike the mean-variance case. For example, we have that

$$V_\pi^\lambda(s) = \underset{a \sim \pi(\cdot|s)}{\mathbb{E}} \left[R_\pi^\lambda(s,a) \right] + \gamma \underset{s' \sim P^\pi(\cdot|s)}{\mathbb{E}} \left[V_\pi^\lambda(s') \right],$$

where $R_\pi^\lambda(s,a) := R(s,a) - \lambda(R(s,a) - J_\pi)^2$ can also be seen as a policy-dependent reward transformation.

In the case where we are using parameterized policies (in the manner described in the beginning of this chapter), the gradient of η_θ [2] with respect to θ was derived in [7] as:

$$\nabla_\theta \eta_\theta = \underset{\substack{s \sim d_{\mu_0, \pi_\theta}(\cdot) \\ a \sim \pi_\theta(\cdot|s)}}{\mathbb{E}} \left[Q_{\pi_\theta}^\lambda(s,a) \nabla_\theta \log \pi_\theta(a|s) \right]. \tag{3.5}$$

[2]We usually write η_θ instead of η_{π_θ} for notational convenience.

Note that this gradient has the same form as the risk-neutral policy gradient (2.11), but here we have the transformed action-value function instead of the normal one. Also note that as V_π^λ is a function of only the states (and not the actions), it satisfies (2.14), and hence can be used as a baseline in the mean-volatility gradient in the same manner that we used baselines in the risk-neutral case (2.15). In other words,

$$\nabla_\theta \eta_\theta = \mathop{\mathbb{E}}_{\substack{s \sim d_{\mu_0, \pi_\theta}(\cdot) \\ a \sim \pi_\theta(\cdot|s)}} \Big[A_{\pi_\theta}^\lambda(s, a) \nabla_\theta \log \pi_\theta(a|s) \Big],$$

where $A_\pi^\lambda(s, a) := Q_\pi^\lambda(s, a) - V_\pi^\lambda(s)$ is the transformed advantage function.

It is clear now that by focusing on the variance of the per-step rewards instead of the returns, the value functions and the policy gradient take convenient forms that are very close to the risk-neutral case. However, adopting risk-neutral algorithms is still not directly possible. For example, if we were to learn V_π^λ, we would need to use the reward transformation R_π^λ. However, such transformation requires J_π, which we do not have access to. Moreover, once the policy is updated, a new transformation would be required. Thus, the simplest way to adopt a risk-neutral policy gradient algorithm for optimizing the mean-volatility is to estimate J_π for the policy at the current iteration and use it to transform the rewards that are used for policy evaluation or improvement. One simple algorithm that follows this scheme is presented next.

3.2 A Monte-Carlo Mean-Volatility Policy Gradient Algorithm

Similar to what was done in [7], we describe a simple actor-only policy gradient algorithm for optimizing the mean-volatility. At each iteration, the algorithm uses a Monte-Carlo simulation to obtain an estimate of the expected return J_π for the current policy. More specifically, the algorithm simulates L episodes each truncated at a fixed horizon of T_J steps, and then averages the returns from these episodes. The pseudo-code for the Monte-Carlo estimation of J_π is provided in Algorithm 3.1.

Note that we normalize the returns using the $(1-\gamma)$ factor in accordance with the way we defined J_π in (3.1). Also note that $\hat{J}$ is not necessarily an unbiased estimate of J_π since we are truncating the return samples. This (small) bias is precisely characterized in the next chapter. Having obtained an estimate of J_π, we use it to transform the rewards of another trajectory and use them to calculate an estimate of the mean-volatility gradient $\nabla_\theta \eta_\theta$ via a trajectory-wise version of (2.13) in the risk-neutral case. To derive this version, we can use (3.5) and (2.12) to write:

Algorithm 3.1 Monte-Carlo-J

```
 1: Input: $\pi, \gamma, L, T_J$
 2: Initialize: $G_0, \dots, G_{L-1} = 0$
 3: for $i = 0, \dots, L-1$ do
 4:     $s_0 \sim \mu_0(\cdot)$
 5:     for $t = 0, \dots, T_J - 1$ do
 6:         $a_t \sim \pi(s_t)$, $s_{t+1} \sim P(\cdot|s_t, a_t)$
 7:         $G_i = G_i + \gamma^t R(s_t, a_t)$
 8:     end for
 9: end for
10: $\hat{J} = \frac{1}{L}\sum_{i=0}^{L-1}(1-\gamma)G_i$
11: Output: $\hat{J}$
```

Algorithm 3.2 Monte-Carlo Mean-Volatility Policy Gradient Algorithm

```
1: Input: Policy Class $\pi_\theta$, $\lambda, \gamma, L, T_J, T, N$
2: Initialize: $\theta_0$
3: for $i = 0, \dots, N-1$ do
4:     $\hat{J}_i$ = Monte-Carlo-J$(\pi_{\theta_i}, \gamma, L, T_J)$
5:     Define $\hat{R}_i^\lambda(s,a) := R(s,a) - \lambda\Big(R(s,a) - \hat{J}_i\Big)^2$
6:     Sample a trajectory $\tau_i := (s_{i,0}, a_{i,0}, \dots, s_{i,T-1}, a_{i,T-1})$ using $\pi_{\theta_i}$.
7:     $\theta_{i+1} = \theta_i + \alpha_i \sum_{t=0}^{T-1} \gamma^t \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \sum_{t'=t}^{T-1} \gamma^{t'-t} \hat{R}_i^\lambda(s_{i,t'}, a_{i,t'})$
8: end for
9: Output: $\theta_N$
```

$$
\begin{aligned}
\nabla_\theta \eta_\theta &= \mathop{\mathbb{E}}_{\substack{s \sim d_{\mu_0,\pi_\theta}(\cdot) \\ a \sim \pi_\theta(\cdot|s)}} \Big[Q_{\pi_\theta}^\lambda(s,a) \nabla_\theta \log \pi_\theta(a|s)\Big] \\
&= (1-\gamma) \mathop{\mathbb{E}}_{s_0 \sim \mu_0(\cdot)} \left[\int_{\mathcal{S}} \int_{\mathcal{A}} \sum_{t=0}^{\infty} \gamma^t p_{\pi_\theta}(s_0 \xrightarrow{t} s) \pi_\theta(a|s) Q_{\pi_\theta}^\lambda(s,a) \nabla_\theta \log \pi_\theta(a|s)\, da\, ds\right] \\
&= (1-\gamma) \mathop{\mathbb{E}}_{s_0 \sim \mu_0(\cdot)} \left[\sum_{t=0}^{\infty} \int_{\mathcal{S}} \int_{\mathcal{A}} p_{\pi_\theta}(s_0 \xrightarrow{t} s) \pi_\theta(a|s) \gamma^t Q_{\pi_\theta}^\lambda(s,a) \nabla_\theta \log \pi_\theta(a|s)\, da\, ds\right] \\
&= (1-\gamma) \mathop{\mathbb{E}}_{s_0 \sim \mu_0(\cdot)} \left[\sum_{t=0}^{\infty} \mathop{\mathbb{E}}_{\substack{s_t \sim p_{\pi_\theta}(s_0 \xrightarrow{t} \cdot) \\ a_t \sim \pi(\cdot|s_t)}} \Big[\gamma^t Q_{\pi_\theta}^\lambda(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)\Big]\right].
\end{aligned}
$$

We use a simulated trajectory to obtain a sample of this form of the gradient, in which we use the sampled (transformed) returns in place of $Q_{\pi_\theta}^\lambda$. The pseudo-code of the full algorithm is provided in Algorithm 3.2.

Although we are following a Monte-Carlo simulation approach, the gradient estimates in Algorithm 3.2 are biased due to multiple reasons. Firstly, we truncate the trajectories at T steps, which can introduce bias if the MDP is not guaranteed to reach a terminal state before then. Moreover, the transformed rewards are biased[3] since we only have one estimate of J_π and it is involved in a squared term. Even if we have two independent estimates of J_π, the transformed rewards will still be biased if these estimates are biased. One can go one step further and consider a variant (more similar to the actor-only algorithm described in [7]) in which we estimate the gradient using the trajectories we used to estimate J_π. The policy update rule (at the k^{th} iteration) for this variant can be written as:

$$\theta_{k+1} = \theta_k + \alpha_k \frac{1}{L} \sum_{i=0}^{L-1} \sum_{t=0}^{T_J-1} \gamma^t \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \sum_{t'=t}^{T-1} \gamma^{t'-t} \hat{R}_k^\lambda(s_{i,t'}, a_{i,t'}).$$

Note that in this case the estimate of J_π and the trajectories used to estimate the gradient are no longer independent.

The main goal of this work is to understand the sample complexity of algorithms like Algorithm 3.2 compared to their risk-neutral counterparts. This will involve understanding how the extra process of estimating J_π affects sample complexity, having in mind the biasedness and dependence issues that we have highlighted. Moreover, we would, in practice, prefer an actor-critic version, where the transformed value function is learned using function approximation. This, as we discussed before, reduces the variance and allows for on-line learning. In that case, we would also need to understand how the inaccuracy of the learned reward transformation affects the prediction accuracy of the critic.

Since the policy gradient for the mean-volatility (3.5) takes a form similar to that of the policy gradient in the risk neutral case (2.11), the main distinguishing features of mean-volatility algorithms will be in the policy evaluation phase. Thus, we focus first on the analysis of the policy evaluation problem, and eventually return to the analysis of a full actor-critic algorithm. The only approach we suggested till now for policy evaluation[4] is the one where we estimate the expected return J_π and use it to transform the rewards, which then enables us to use any standard policy evaluation method. We can call this approach *the direct method.* The general scheme of the direct method is summarized in Algorithm 3.3. This scheme can be combined with different sampling strategies as discussed before. Namely, we can obtain two independent estimates of J_π to be used in the reward transformation, and we can also use for policy evaluation the same data that was used to obtain $\hat{J}$. In the next section, we describe another approach for policy evaluation, which we call *the factored method*, where we do not use $\hat{J}$ in any reward transformation.

[3]Note that the reward function is a deterministic function of state-action pairs; the stochasticity considered here is due to the sampling process used to estimate J_π.

[4]Although we did not use a critic in the presented algorithm, providing estimates of $Q^\lambda_{\pi_\theta}$ at the encountered state-action pairs using sampled (transformed) returns could be seen as a rudimentary form of policy evaluation.

Algorithm 3.3 Mean-Volatility Policy Evaluation: The Direct Method

```
1: Input: a policy $\pi$, risk-aversion parameter $\lambda$, discount factor $\gamma$.
2: Obtain $\hat{J}$ as an estimate of $J_\pi$.
3: Define the reward transformation $\hat{R}^\lambda(s,a) := R(s,a) - \lambda\Big(R(s,a) - \hat{J}\Big)^2$
4: Obtain $\hat{V}^\lambda$ as an estimate of $V_\pi^\lambda$ using any algorithm for learning the risk-
   neutral state-value function while transforming the rewards using $\hat{R}^\lambda$.
5: Output: $\hat{V}^\lambda$
```

3.3 The Factored Method for Mean-Volatility Policy Evaluation

Before describing the factored method, we briefly discuss an alternative method, which we hinted at before, for optimizing the mean volatility. This method, introduced in [37], is called *Mean-Variance Policy Iteration* (MVPI). We can start by recalling that for a random variable X, $Var(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$. We can apply this for the reward volatility term in the mean volatility objective, thus yielding:

$$\begin{aligned}\eta_\pi &= J_\pi - \lambda\nu_\pi^2\\ &= \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi}(\cdot)\\ a\sim\pi(\cdot|s)}}[R(s,a)] - \lambda \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi}(\cdot)\\ a\sim\pi(\cdot|s)}}\Big[(R(s,a)-J_\pi)^2\Big]\\ &= \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi}(\cdot)\\ a\sim\pi(\cdot|s)}}[R(s,a)] - \lambda \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi}(\cdot)\\ a\sim\pi(\cdot|s)}}\Big[R(s,a)^2\Big] + \lambda \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi}(\cdot)\\ a\sim\pi(\cdot|s)}}[R(s,a)]^2.\end{aligned}$$

They then use the Fenchel duality to write

$$\eta_\pi = \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi}(\cdot)\\ a\sim\pi(\cdot|s)}}[R(s,a)] - \lambda \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi}(\cdot)\\ a\sim\pi(\cdot|s)}}\Big[R(s,a)^2\Big] + \lambda \max_y \left(2 \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi}(\cdot)\\ a\sim\pi(\cdot|s)}}[R(s,a)]y - y^2\right).$$

Thus, they aim to obtain $\max_{\pi,y}\eta_{\pi,y}$, where

$$\eta_{\pi,y} := \mathop{\mathbb{E}}_{\substack{s\sim d_{\mu_0,\pi}(\cdot)\\ a\sim\pi(\cdot|s)}}\Big[R(s,a) - \lambda R(s,a)^2 + 2\lambda R(s,a)y\Big] - \lambda y^2.$$

To optimize this objective function, they propose a *block cyclic coordinate ascent* (BCCA) approach, where they alternate between fixing either π or y and optimizing over the other. When fixing π, the optimal choice for y is in fact J_π, which still needs to be estimated. On the other hand, when y is fixed, the optimal π is the solution to the risk-neutral control problem with reward function $R(s,a) - \lambda R(s,a)^2 + 2\lambda R(s,a)y$, where any reinforcement learning algorithm can be used since this reward transformation is kept fixed even when the policy gets updated during this policy optimization phase.

One of the merits of MVPI is that the term y^2 (or J^2) is not involved in the reward transformation. We can attempt to perform a similar factorization for the transformed value function V_π^λ. To this end, we can define $d_\pi(\cdot|s)$ as the discounted state distribution when starting from state s. That is,

$$d_\pi(\cdot|s) := (1-\gamma)\sum_{t=0}^{\infty}\gamma^t p_{\pi_\theta}(s \xrightarrow{t} \cdot). \tag{3.6}$$

We can then, analogous to (3.3), write V_π^λ as:

$$\begin{aligned} V_\pi^\lambda(s') &= \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d_\pi(\cdot|s')\\ a\sim\pi(\cdot|s)}}\Big[R(s,a) - \lambda(R(s,a)-J_\pi)^2\Big] \\ &= \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d_\pi(\cdot|s')\\ a\sim\pi(\cdot|s)}}[R(s,a)] - \frac{\lambda}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d_\pi(\cdot|s')\\ a\sim\pi(\cdot|s)}}\Big[(R(s,a)-J_\pi)^2\Big]. \end{aligned}$$

Unfortunately, the second term, unlike the one in η_π, is not a variance term since, in general,

$$J_\pi \neq \mathop{\mathbb{E}}_{\substack{s\sim d_\pi(\cdot|s')\\ a\sim\pi(\cdot|s)}}[R(s,a)].$$

This means that we cannot use the relation between the variance and the second moment. Still, we can factorize the squared term to obtain that

$$\begin{aligned} V_\pi^\lambda(s') = \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d_\pi(\cdot|s')\\ a\sim\pi(\cdot|s)}}[R(s,a)] - \frac{\lambda}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d_\pi(\cdot|s')\\ a\sim\pi(\cdot|s)}}\Big[R(s,a)^2\Big] \\ + \frac{2\lambda J_\pi}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d_\pi(\cdot|s')\\ a\sim\pi(\cdot|s)}}[R(s,a)] - \frac{\lambda J_\pi^2}{1-\gamma}. \end{aligned}$$

We can then use the definition of the state-value function V^π in (1.2) and the definition of $d_\pi(\cdot|s)$ in (3.6) to conclude that

$$V_\pi^\lambda(s) = (1+2\lambda J_\pi)V^\pi(s) - \lambda M^\pi(s) - \frac{\lambda}{1-\gamma}J_\pi^2, \tag{3.7}$$

where we call $M^\pi : \mathcal{S} \to \mathbb{R}$ the *second moment value function*[5], which is defined as follows:

$$M^\pi(s) := \mathop{\mathbb{E}}_{\substack{a_t\sim\pi(\cdot|s_t)\\ s_{t+1}\sim P(\cdot|s_t,a_t)}}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)^2 \middle| s_0 = s\right]. \tag{3.8}$$

Squaring the rewards can be seen as a deterministic (policy-independent) reward transformation. Thus, M^π can be learned by adapting any algorithm that can be used for learning V^π. Note also that any accuracy guarantees (e.g. finite-time bounds) on V^π can be adapted for M^π; we would just need to consider that the range of values of the step-rewards is different.

[5]It is the second moment of the step reward $R(s',a')$ (where $s' \sim d_\pi(\cdot|s)$ and $a' \sim \pi(\cdot|s')$), not of the return when starting from s.

Algorithm 3.4 Mean-Volatility Policy Evaluation: The Factored Method

```
1: Input: a policy $\pi$, risk-aversion parameter $\lambda$, discount factor $\gamma$.
2: Obtain $\hat{J}$ as an estimate of $J_\pi$.
3: Obtain $\hat{V}$ as an estimate of $V^\pi$ using any algorithm for learning the risk-neutral state-value function.
4: Obtain $\hat{M}$ as an estimate of $M^\pi$ using any algorithm for learning the risk-neutral state-value function while squaring the step rewards.
5: Set $\hat{V}^\lambda(s) = (1 + 2\lambda\hat{J})\hat{V}(s) - \lambda\hat{M}(s) - \frac{\lambda}{1-\gamma}\hat{J}^2 \quad \forall s \in \mathcal{S}$.
6: Output: $\hat{V}^\lambda$
```

In the factored method, we use (3.7) to estimate V_π^λ. This means that we would need to estimate V^π, M^π, and J_π separately, and then combine them using (3.7) to obtain an estimate of V_π^λ. Note that this approach is not too different from the approach adopted in [5] for estimating the variance of the reward to go. However, the algorithm in our case is simpler. This is mainly because, although we still need to learn J_π, we do not use it (or the value function) in any reward transformation. In fact, the only reward transformation required is the squaring of the rewards that we perform when estimating M^π. This transformation, however, is exact (it does not depend on any estimates) and does not depend on the policy that is being evaluated. This is unlike the second moment of the return (that we defined in (2.20)), which requires a reward transformation that, in addition to the squared rewards, involves the risk-neutral value function of the policy (see (2.21)), which introduces extra complications.

The general scheme of the factored approach is summarized in Algorithm 3.4. As in the direct method, the choice of the sampling scheme is relevant since the multiplication of estimates (as in $\hat{J}^2$ and the multiplication of $\hat{J}$ and $\hat{V}$) can introduce bias if they are learnt from the same data. We will see examples of this later on. One interesting thing about the factored method is that we can derive generic error bounds on $\hat{V}^\lambda$ given error bounds on $\hat{M}$ and $\hat{V}$, regardless of what method was used to learn them. This is indeed the subject of the next chapter. On the other hand, analyzing the direct method depends on the used policy evaluation method. Later on, we will analyze a full actor-critic algorithm that uses linear mini-batch TD(0) (see Subsection 2.1.3) as a critic that learns the transformed value function using the direct method.

Chapter 4

A General Error Bound for the Factored Method

In this chapter, we derive a simple general bound on the prediction error of the learned estimates of the transformed value function V_π^λ when using the factored method (Algorithm 3.4). The bound is general in the sense that it does not require a specific method for learning the state-value function V^π or the second moment value function M^π, it only requires that we are given error bounds (in a certain format) for the learned estimates of the two functions. Throughout the chapter, we develop bounds on a number of quantities, and eventually combine them to obtain the full bound. We end the chapter by providing an expanded form of the bound when V^π and M^π are learned using least-squares temporal difference, where we use the finite-time bounds derived in [38].

We assume that we are still operating in the MDP $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu_0 \rangle$ defined in beginning of the previous chapter. Remember also our assumption that $|R(s,a)| \leq R_{\max}\ \forall (s,a) \in S \times A$. Moreover, since we are focusing only on the policy evaluation problem, we shall drop the dependence on π from the notation for clarity. This means that we will use V^λ, M, V, and J instead of V_π^λ, M^π, V^π, and J^π.

4.1 The Problem

As we discussed in the previous chapter, the idea of the factored method is to find estimates of V, M, and J separately and combine them using (3.7) to form an estimate of V^λ. That is, if $\hat{V}$, $\hat{M}$, and $\hat{J}$ are our estimates of V, M, and J, then our estimate $\hat{V}^\lambda$ of the transformed value function at any state s is defined according to:

$$\hat{V}^\lambda(s) = (1 + 2\lambda\hat{J})\hat{V}(s) - \lambda\hat{M}(s) - \frac{\lambda}{1-\gamma}\hat{J}^2. \tag{4.1}$$

In the following, we do not necessarily assume that $\hat{J}$ and $\hat{V}$ are independent; our analysis holds even if they are learned from the same data. Moreover, we assume that

$\hat{J}$ is a Monte-Carlo estimate of J (see Algorithm 3.1), and we rely on only one such estimate. In our analysis, we treat $\hat{V}^\lambda$, $\hat{M}$, $\hat{V}$, and $\hat{J}$ as random variables, where their stochasticity comes from the sampling processes that are used to learn them.

Our goal is to derive bounds on the error of $\hat{V}^\lambda$. However, we first need to define a notion of accuracy according to which we can evaluate our estimate. Suppose μ is a probability measure over $\mathcal{S}$ and f is a bounded measurable function with $\mathcal{S}$ as its domain, we can then define the following norm:

$$\|f\|_\mu^2 := \int_{\mathcal{S}} f(s)^2 \mu(ds). \tag{4.2}$$

Accordingly, our aim is to bound $\|V^\lambda - \hat{V}^\lambda\|_\mu$, where μ is usually chosen to be the stationary state distribution of the Markov chain (if it admits one). Moreover, for any two functions of the states f_1 and f_2, we define the sum $f_1 + f_2$ as a function with the obvious meaning that $(f_1 + f_2)(s) := f_1(s) + f_2(s)$ for any $s \in \mathcal{S}$. And for any $a \in \mathbb{R}$, $(af_1)(s) := af_1(s)$ for any $s \in \mathcal{S}$[1].

A starting point is the following simple passage concerning the difference between V^λ and $\hat{V}^\lambda$ at some state s:

$$\begin{aligned}
&(V^\lambda - \hat{V}^\lambda)(s) \\
&= \left[(1+2\lambda J)V(s) - (1+2\lambda\hat{J})\hat{V}(s)\right] - \lambda\left[M(s) - \hat{M}(s)\right] - \frac{\lambda}{1-\gamma}\left[J^2 - \hat{J}^2\right] \\
&= \left[V(s) - \hat{V}(s)\right] + 2\lambda\left[JV(s) - \hat{J}\hat{V}(s)\right] + \lambda\left[\hat{M}(s) - M(s)\right] + \frac{\lambda}{1-\gamma}\left[\hat{J}^2 - J^2\right] \\
&= \left(V - \hat{V}\right)(s) + 2\lambda\left(JV - \hat{J}\hat{V}\right)(s) + \lambda\left(\hat{M} - M\right)(s) + \frac{\lambda}{1-\gamma}\left(\hat{J}^2 - J^2\right)(s),
\end{aligned}$$

where $\left(\hat{J}^2 - J^2\right)(s) = \left(\hat{J}^2 - J^2\right)\ \forall s \in \mathcal{S}$. We can then write the following:

$$\begin{aligned}
\|V^\lambda - \hat{V}^\lambda\|_\mu &= \left\|\left(V - \hat{V}\right) + 2\lambda\left(JV - \hat{J}\hat{V}\right) + \lambda\left(\hat{M} - M\right) + \frac{\lambda}{1-\gamma}\left(\hat{J}^2 - J^2\right)\right\|_\mu \\
&\le \|V - \hat{V}\|_\mu + 2\lambda\|JV - \hat{J}\hat{V}\|_\mu + \lambda\|\hat{M} - M\|_\mu + \frac{\lambda}{1-\gamma}|\hat{J}^2 - J^2|, \qquad (4.3)
\end{aligned}$$

where we have applied the triangle inequality three times, and used the fact that $\|f\|_\mu = |a|$ when $f(s) = a\ \forall s \in \mathcal{S}$. This then reduces our task to bounding the four terms in (4.3). We assume that we are already provided bounds on the first and the third terms depending on the algorithms that were used to learn them. We will state this assumption more explicitly later. We focus next on analyzing the second term.

4.2 Bounding the Cross Term

We focus now on the term $\|JV - \hat{J}\hat{V}\|_\mu$. That is, we need to bound the $\|\cdot\|_\mu$ norm of the difference between the true value function V multiplied (at each state) by the true

[1]By combining the two rules, we get that $(f_1 + (-f_2))(s) = f_1(s) - f_2(s) := (f_1 - f_2)(s)$ $\forall s \in \mathcal{S}$.

expected return J and our estimate of the value function $\hat{V}$ multiplied by our estimate of the expect return $\hat{J}$. Without making any assumptions on the method via which we learn $\hat{V}$ or $\hat{J}$, nor assuming that they are learned from different data, the following lemma provides a bound on $\|JV - \hat{J}\hat{V}\|_\mu$ via an elementary decomposition.

Lemma 4.1. *Assuming that* $\|\hat{V}\|_\mu \leq \frac{R_{\max}}{1-\gamma}$ *and that* $|\hat{J}| \leq R_{\max}$*, then the following holds:*

$$\|JV - \hat{J}\hat{V}\|_\mu \leq R_{\max}\left(\frac{|J-\hat{J}|}{1-\gamma} + \|V-\hat{V}\|_\mu\right).$$

Proof.

$$\begin{aligned}
\|JV - \hat{J}\hat{V}\|_\mu &= \|JV - \hat{J}V + \hat{J}V - \hat{J}\hat{V}\|_\mu \\
&\leq \|JV - \hat{J}V\|_\mu + \|\hat{J}V - \hat{J}\hat{V}\|_\mu \\
&= |J-\hat{J}|\,\|V\|_\mu \ + \ |\hat{J}|\,\|V-\hat{V}\|_\mu \\
&\leq |J-\hat{J}|\,\frac{R_{\max}}{1-\gamma} \ + \ R_{\max}\,\|V-\hat{V}\|_\mu \\
&= R_{\max}\left(\frac{|J-\hat{J}|}{1-\gamma} + \|V-\hat{V}\|_\mu\right),
\end{aligned}$$

where the first inequality is an application of the triangle inequality. □

The assumptions made in the lemma are not restricting in any way. The assumption on $\hat{V}$ can be satisfied if $|\hat{V}(s)| \leq \frac{R_{\max}}{1-\gamma}$ $\forall s \in \mathcal{S}$. Even if this does not hold, we can always define a truncated version of $\hat{V}$ defined as follows ($\forall s \in \mathcal{S}$):

$$\tilde{V}(s) = \begin{cases} \hat{V}(s) & \text{if } |\hat{V}(s)| \leq \frac{R_{\max}}{1-\gamma}, \\ \operatorname{sign}(\hat{V}(s))\frac{R_{\max}}{1-\gamma} & \text{otherwise.} \end{cases}$$

We can then use $\tilde{V}$ in place of $\hat{V}$ without incurring any loss (according to $\|V - \cdot\|_\mu$ or any sensible notion of accuracy) since the true value function V satisfies $|V(s)| \leq \frac{R_{\max}}{1-\gamma}$ $\forall s \in \mathcal{S}$. We can also make an analogous argument for $\hat{J}$, but for our interest, it suffices to see that $|\hat{J}| \leq R_{\max}$ automatically holds if $\hat{J}$ is learnt using Algorithm 3.1.

Lemma 4.1 states that we can bound $\|JV - \hat{J}\hat{V}\|_\mu$ in terms of $|J-\hat{J}|$ and $\|V-\hat{V}\|_\mu$. As we mentioned before, we will assume later that we have a certain form of bound on $\|V-\hat{V}\|_\mu$. For now, we will focus on bounding $|J-\hat{J}|$, along with $|\hat{J}^2 - J^2|$ from (4.3), when $\hat{J}$ is learnt using Algorithm 3.1.

4.3 Analysing the Monte-Carlo Estimation of the Expected Return

The basic idea of Algorithm 3.1 is to average the (normalized) returns from L simulated trajectories each truncated at T_J steps. The goal in this section is to derive finite-time bounds on $|J - \hat{J}|$ and $|\hat{J}^2 - J^2|$. That is, we want to bound these quantities in terms of the number of simulated trajectories and the horizon length of the trajectories. We can start by laying out a summary of all the relevant quantities for the analysis:

- L: number of simulated episodes.
- T_J: number of simulated steps in each episode.
- $G_{0:T_J-1}$: a random variable representing the discounted sum of rewards from the beginning of a trajectory up to time $T_J - 1$ multiplied by a factor of $1-\gamma$. That is:
$$G_{0:T_J-1} := (1-\gamma) \sum_{t=0}^{T_J-1} \gamma^t R(s_t, a_t).$$
We denote its expected value by
$$\overline{G}_{0:T_J-1} := (1-\gamma) \mathop{\mathbb{E}}_{\substack{s_{t+1}\sim P(\cdot|s_t,a_t)\\ a_t\sim\pi(\cdot|s_t)}} \left[\sum_{t=0}^{T_J-1} \gamma^t R(s_t, a_t) \middle| s_0 \sim \mu(\cdot) \right].$$
Note that we have that (with probability 1) $|G_{0:T_J-1}| \leq (1-\gamma^{T_J}) R_{max} \leq R_{max}$.
- $G_i : i = 0, \ldots, L-1$: i.i.d.[2] samples of $G_{0:T_J-1}$ corresponding to each simulated trajectory.
- ζ_i: i^th^ central moment[3] of $G_{0:T_J-1}$.
- $\hat{J} = \frac{1}{L}\sum_{i=0}^{L-1} G_i$.
- $G_{T_J:\infty}$: a random variable representing the discounted sum of rewards collected starting from time T_J onward multiplied by a factor of $\gamma^{T_J}(1-\gamma)$. That is:
$$G_{T_J:\infty} := (1-\gamma)\gamma^{T_J} \sum_{t=0}^{\infty} \gamma^t R(s_{t+T_J}, a_{t+T_J}) = (1-\gamma) \sum_{t=T_J}^{\infty} \gamma^t R(s_t, a_t).$$
Its expected value can then be defined as:
$$\overline{G}_{T_J:\infty} := (1-\gamma) \mathop{\mathbb{E}}_{\substack{s_{t+1}\sim P(\cdot|s_t,a_t)\\ a_t\sim\pi(\cdot|s_t)}} \left[\sum_{t=T_J}^{\infty} \gamma^t R(s_t, a_t) \middle| s_{T_J} \sim \int_S p_\pi(s_0 \xrightarrow{T_J} \cdot)\mu(ds_0) \right].$$
This way, we have that $J = \overline{G}_{0:T_J-1} + \overline{G}_{T_J:\infty}$. Also note that $|G_{T_J:\infty}| \leq \gamma^{T_J} R_{max} \leq R_{max}$.

The following (basic) lemma provides upper bounds on the second, third, and fourth central moments of $G_{0:T_J-1}$. We will need these bounds later in our analysis.

[2]Short for *independent and identically distributed.*

[3]For a random variable X, its i^th^ central moment is $\mathbb{E}\Big[(X - \mathbb{E}[X])^i\Big]$. Note that the second central moment of X is its variance.

Lemma 4.2. *With ζ_i denoting the i^{th} central moment of $G_{0:T_J-1}$, we have*

i. $\zeta_2 \leq R_{max}^2$.

ii. $|\zeta_3| \leq \frac{4\sqrt{3}}{9} R_{max}^3 \leq R_{max}^3$.

iii. $\zeta_4 \leq \frac{4}{3} R_{max}^4 \leq 2R_{max}^4$.

Proof. For a random variable X upper-bounded by M and lower bounded by m, with μ_i denoting its ith central moment, we have by Popoviciu's inequality that

$$\mu_2 \leq \frac{(M-m)^2}{4}.$$

Thus, we have that

$$\zeta_2 \leq \frac{(2R_{max})^2}{4} = R_{max}^2,$$

since we can take $M = R_{max}$ and $m = -R_{max}$, proving the first item. For the second item we have from Theorem (2.3) in [39] that

$$|\mu_3| \leq \frac{(M-m)^3}{6\sqrt{3}}.$$

Which means that in our case we shall have that

$$|\zeta_3| \leq \frac{(2R_{max})^3}{6\sqrt{3}} = \frac{8R_{max}^3}{6\sqrt{3}} = \frac{4\sqrt{3}}{9} R_{max}^3 \leq R_{max}^3.$$

Finally, for the third item, Theorem (2.1) in [39] states that

$$\mu_4 \leq \frac{(M-m)^4}{12}.$$

And for us,

$$\zeta_4 \leq \frac{(2R_{max})^4}{12} \leq \frac{4}{3} R_{max}^4 \leq 2R_{max}^4.$$

□

4.3.1 Bounding the Error on the Estimated Expected Reward

One can notice from Algorithm 3.1 that $\hat{J}$ is not necessarily unbiased. Which means that, in general, $\mathbb{E}[\hat{J}] \neq J$. This is because we truncate the trajectories, and consequently, our samples of the return are not unbiased. A generally more important property of an estimator is consistency. A consistent estimator is an estimator that (whatever the true value its trying to estimate is) can achieve any given level of accuracy with any given level of confidence provided that the number of samples is large enough [40]. We will indeed show that $\hat{J}$, learned using Algorithm 3.1, enjoys this property.

We can start with the following decomposition:

$$\begin{aligned} |J - \hat{J}| &= |J - \mathbb{E}[\hat{J}] + \mathbb{E}[\hat{J}] - \hat{J}| \\ &\leq |J - \mathbb{E}[\hat{J}]| + |\hat{J} - \mathbb{E}[\hat{J}]|. \end{aligned} \tag{4.4}$$

The first term in (4.4) represents the bias of $\hat{J}$, which we can control by adjusting the length of the trajectories as stated in the following simple lemma, which we will refer to again in the future.

Lemma 4.3. *The following holds when $\hat{J}$ is learned using Algorithm* 3.1*:*

$$|J - \mathbb{E}[\hat{J}]| \leq \gamma^{T_J} R_{max}.$$

Proof. Since $\hat{J}$ is a sample average of instances of $G_{0:T_J-1}$, its expected value is the same as that of $G_{0:T_J-1}$, which is $\overline{G}_{0:T_J-1}$. Moreover, we remarked earlier that $J = \overline{G}_{0:T_J-1} + \overline{G}_{T_J:\infty}$, this then means that

$$|J - \mathbb{E}[\hat{J}]| = |J - \overline{G}_{0:T_J-1}| = |\overline{G}_{T_J:\infty}| \leq \gamma^{T_J} R_{max},$$

which concludes the proof. □

The second term in (4.4) represent how far a sample average (as in $\hat{J}$) is far from its expected value. The following proposition bounds this term using a simple concentration inequality, and combines that with the result in Lemma 4.3 to obtain the desired bound on $|J - \hat{J}|$.

Proposition 4.1. *Assuming $\hat{J}$ is learned using Algorithm 3.1, we have, with probability at least* $1 - \delta_j$*, that*

$$|J - \hat{J}| \leq R_{max} \left(\gamma^{T_J} + \sqrt{\frac{2 \log\left(\frac{2}{\delta_j}\right)}{L}} \right).$$

Proof. With $\bar{X}$ denoting the empirical mean of n independent random variables $X_1, \ldots, X_n$ each bounded by the interval $[a, b]$, the Hoeffding inequality [41] states that[4]

$$P\left(\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \geq t\right) \leq 2 \exp\left(\frac{-2n^2t^2}{n(b-a)^2}\right).$$

This translates in our case to

$$P\left(\left|\hat{J} - \mathbb{E}[\hat{J}]\right| \geq t\right) \leq 2 \exp\left(\frac{-2L^2t^2}{L(R_{max} - (-R_{max}))^2}\right) = 2 \exp\left(\frac{-Lt^2}{2R_{max}^2}\right),$$

[4]Note that t here has nothing to do with time.

which implies that

$$\left|\hat{J} - \mathbb{E}[\hat{J}]\right| \leq R_{max}\sqrt{\frac{2\log\left(\frac{2}{\delta_j}\right)}{L}}$$

holds with probability at least[5] $1-\delta_j$. Combining this with inequality (4.4) and Lemma 4.3, we can state, with probability at least $1-\delta_j$, that

$$\begin{aligned}|J - \hat{J}| &\leq \gamma^{T_J} R_{max} + R_{max}\sqrt{\frac{2\log\left(\frac{2}{\delta_j}\right)}{L}} \\ &= R_{max}\left(\gamma^{T_J} + \sqrt{\frac{2\log\left(\frac{2}{\delta_j}\right)}{L}}\right),\end{aligned}$$

which is the required statement. □

One can see now that we can achieve any level of accuracy (i.e. $|J-\hat{J}| \leq \epsilon$ for any $\epsilon > 0$) with any level of confidence (i.e. with probability at least $1-\delta_j$, for any $\delta_j \in (0,1]$) by making L and T_J large enough. This makes $\hat{J}$ a consistent estimator.

4.3.2 Bounding the Error on the Estimated Squared Expected Reward

The next challenge is to bound the term $|\hat{J}^2 - J^2|$, which appears in (4.3). This term arises since there is a J^2 term in (3.7) which we, as indicated in (4.1), estimate by squaring our expected return estimator $\hat{J}$. Similar to what we have done for $\hat{J}$, we can start with the following decomposition:

$$\begin{aligned}|J^2 - \hat{J}^2| &= |J^2 - \mathbb{E}[\hat{J}^2] + \mathbb{E}[\hat{J}^2] - \hat{J}^2| \\ &\leq |J^2 - \mathbb{E}[\hat{J}^2]| + |\mathbb{E}[\hat{J}^2] - \hat{J}^2|. \qquad (4.5)\end{aligned}$$

Where the first term represents the bias of the estimator $\hat{J}^2$, and the second term represents how far $\hat{J}^2$ is from its expected value. The following lemma provides a bound on the first term.

Lemma 4.4. *With $\hat{J}$ learned using Algorithm* 3.1, *we have that*

$$|J^2 - \mathbb{E}[\hat{J}^2]| \leq R_{max}^2\left(2\gamma^{T_J} + \frac{1}{L}\right).$$

Proof. Recall that for a random variable X, $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$. Which means that $\mathbb{E}[\hat{J}^2] = \mathbb{E}[\hat{J}]^2 + \mathrm{Var}(\hat{J})$. Consequently,

$$|J^2 - \mathbb{E}[\hat{J}^2]| = |J^2 - \mathbb{E}[\hat{J}]^2 + \mathrm{Var}(\hat{J})|$$

[5]Note that $0 < \delta_j \leq 1$.

$$\begin{aligned}
&\overset{(1)}{=} \left| J^2 - \mathbb{E}[\hat{J}]^2 + \frac{\zeta_2}{L} \right| \\
&\leq |J^2 - \mathbb{E}[\hat{J}]^2| + \frac{\zeta_2}{L} \\
&\overset{(2)}{\leq} |J^2 - \mathbb{E}[\hat{J}]^2| + \frac{R_{max}^2}{L},
\end{aligned}$$

where (1) holds since $\hat{J}$ is an empirical mean of L instances of $G_{0:T_J-1}$, and thus $\text{Var}(\hat{J}) = \frac{\text{Var}(G_{0:T_J-1})}{L} = \frac{\zeta_2}{L}$. The step labelled (2) then follows by Lemma 4.2.i. To proceed, remember that $\mathbb{E}[\hat{J}] = \overline{G}_{0:T_J-1}$, and that $J = \overline{G}_{0:T_J-1} + \overline{G}_{T_J:\infty}$. This means that

$$\begin{aligned}
|J^2 - \mathbb{E}[\hat{J}^2]| &\leq \left|(\overline{G}_{0:T_J-1} + \overline{G}_{T_J:\infty})^2 - \overline{G}_{0:T_J-1}^2\right| + \frac{R_{max}^2}{L} \\
&= \left|\overline{G}_{T_J:\infty}(2\overline{G}_{0:T_J-1} + \overline{G}_{T_J:\infty})\right| + \frac{R_{max}^2}{L} \\
&\leq \left|\overline{G}_{T_J:\infty}\right|(2\left|\overline{G}_{0:T_J-1}\right| + \left|\overline{G}_{T_J:\infty}\right|) + \frac{R_{max}^2}{L} \\
&\leq \gamma^{T_J} R_{max} \left(2(1-\gamma^{T_J})R_{max} + \gamma^{T_J} R_{max}\right) + \frac{R_{max}^2}{L} \\
&= \gamma^{T_J}(2-\gamma^{T_J})R_{max}^2 + \frac{R_{max}^2}{L} \\
&\leq 2\gamma^{T_J} R_{max}^2 + \frac{R_{max}^2}{L} \\
&= R_{max}^2\left(2\gamma^{T_J} + \frac{1}{L}\right),
\end{aligned}$$

where we used in the second inequality the fact that $|\overline{G}_{0:T_J-1}| \leq (1-\gamma^{T_J})R_{max}$ and that $|\overline{G}_{T_J:\infty}| \leq \gamma^{T_J} R_{max}$. □

The previous lemma states that the bias of $\hat{J}^2$ depends on the bias of $\hat{J}$ (as in Lemma 4.3, but with a larger constant) and the variance of $\hat{J}$, which can be reduced by using more trajectories. We now turn to the second term in (4.5), which, as mentioned before, conveys how far $\hat{J}^2$ is from its expected value. Unlike $\hat{J}$, $\hat{J}^2$ is not an average (nor a sum) of a number of independent random variables, and thus we cannot use the Hoeffding inequality. However, we can use another concentration inequality called the Chebyshev inequality. For a random variable X with mean μ and variance σ^2, the Chebyshev inequality [40] states that

$$P\left(|X - \mu| \geq c\right) \leq \frac{\sigma^2}{c^2}.$$

This can also be used to state, with probability at least $1 - \delta$, that[6]

$$|X - \mu| \leq \sqrt{\frac{\sigma^2}{\delta}}.$$

[6]Note that $0 < \delta \leq 1$.

Applying this to our case, we can state, with probability at least $1 - \delta_{j^2}$, that

$$|\,\mathbb{E}[\hat{J}^2] - \hat{J}^2| \leq \sqrt{\frac{\text{Var}(\hat{J}^2)}{\delta_{j^2}}}. \tag{4.6}$$

But what can we say about the variance of $\hat{J}^2$? The following lemma provides one answer.

Lemma 4.5. *For a generic random variable X with mean $\mathbb{E}[X]$ and sample mean $\hat{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$, where $X_1 \ldots X_N$ are i.i.d. copies of X, we have that*

$$Var[\hat{X}^2] = 4\,\mathbb{E}[X]^2\frac{\mu_2}{N} + \frac{2\mu_2^2 + 4\mu_3\,\mathbb{E}[X]}{N^2} + \frac{\mu_4 - 3\mu_2^2}{N^3},$$

where μ_i is X's ith central moment defined as: $\mu_i = \mathbb{E}[(X - \mathbb{E}[X])^i]$.

Proof.

$$\begin{aligned}
\text{Var}[\hat{X}^2] &= \mathbb{E}[\hat{X}^4] - \mathbb{E}[\hat{X}^2]^2 \\
&= \mathbb{E}[\hat{X}^4] - (\mathbb{E}[X]^2 + \text{Var}[\hat{X}])^2 \\
&= \mathbb{E}[\hat{X}^4] - \mathbb{E}[X]^4 - 2\,\mathbb{E}[X]^2\text{Var}[\hat{X}] - \text{Var}[\hat{X}]^2 \\
&= \mathbb{E}[\hat{X}^4] - \mathbb{E}[X]^4 - 2\,\mathbb{E}[X]^2\frac{\text{Var}[X]}{N} - \frac{\text{Var}[X]^2}{N^2} \\
&= \mathbb{E}[\hat{X}^4] - \mathbb{E}[X]^4 - 2\,\mathbb{E}[X]^2\frac{\mu_2}{N} - \frac{\mu_2^2}{N^2}
\end{aligned}$$

From [42], we have:

$$\mathbb{E}[\hat{X}^4] = \mathbb{E}[X]^4 + 6\,\mathbb{E}[X]^2\frac{\mu_2}{N} + \frac{3\mu_2^2 + 4\mu_3\,\mathbb{E}[X]}{N^2} + \frac{\mu_4 - 3\mu_2^2}{N^3}.$$

The result follows by plugging this back in the previous equation. □

The following proposition uses the lemma above, along with (4.5) and Lemma 4.4, to derive the desired bound on $|J^2 - \hat{J}^2|$.

Proposition 4.2. *Assuming $\hat{J}$ is learned using Algorithm 3.1, we have, with probability at least $1 - \delta_{j^2}$, that*

$$|J^2 - \hat{J}^2| \leq 2R_{max}^2\left(\gamma^{T_J} + \sqrt{\frac{\left(\frac{15}{\delta_{j^2}}\right)}{L}}\right).$$

Proof. We can start by applying Lemma 4.5 to our case[7]:

$$\text{Var}(\hat{J}^2) = 4\overline{G}_{0:T_J-1}^2\frac{\zeta_2}{L} + \frac{2\zeta_2^2 + 4\zeta_3\overline{G}_{0:T_J-1}}{L^2} + \frac{\zeta_4 - 3\zeta_2^2}{L^3}.$$

[7]Here X is $G_{0:T_J-1}$, and $\hat{X}$ is $\hat{J}$.

$$\begin{aligned}
&\overset{(1)}{\leq} 4\overline{G}^2_{0:T_J-1}\frac{\zeta_2}{L} + \frac{2\zeta_2^2 + 4|\zeta_3||\overline{G}_{0:T_J-1}|}{L^2} + \frac{\zeta_4 + 3\zeta_2^2}{L^3}. \\
&\overset{(2)}{\leq} 4R^2_{max}\frac{\zeta_2}{L} + \frac{2\zeta_2^2 + 4|\zeta_3|R_{max}}{L^2} + \frac{\zeta_4 + 3\zeta_2^2}{L^3},
\end{aligned}$$

where we note in (1) that ζ_2, ζ_4, and L are non-negative. And (2) follows since $|\overline{G}_{0:T_J-1}| \leq R_{max}$. We can then apply Lemma 4.2 on the central moments in the last expression to get that

$$\begin{aligned}
\mathrm{Var}(\hat{J}^2) &\leq 4\frac{R^4_{max}}{L} + \frac{2R^4_{max} + 4R^4_{max}}{L^2} + \frac{2R^4_{max} + 3R^4_{max}}{L^3} \\
&\leq R^4_{max}\left(\frac{4}{L} + \frac{6}{L^2} + \frac{5}{L^3}\right) \\
&\leq \frac{15R^4_{max}}{L},
\end{aligned}$$

where the last inequality holds since $L \geq 1$. We can then combine this with (4.6) to get, with probability at least $1 - \delta_{j^2}$, that

$$\begin{aligned}
|\,\mathbb{E}[\hat{J}^2] - \hat{J}^2| &\leq \sqrt{\frac{15R^4_{max}}{\delta_{j^2}L}} \\
&\leq R^2_{max}\sqrt{\frac{\left(\frac{15}{\delta_{j^2}}\right)}{L}}.
\end{aligned}$$

Finally, we can combine this last result with inequality (4.5) and Lemma 4.4 to state, with probability at least $1 - \delta_{j^2}$, that

$$\begin{aligned}
|J^2 - \hat{J}^2| &\leq R^2_{max}\left(2\gamma^{T_J} + \frac{1}{L}\right) + R^2_{max}\sqrt{\frac{\left(\frac{15}{\delta_{j^2}}\right)}{L}} \\
&= R^2_{max}\left(2\gamma^{T_J} + \frac{1}{L} + \sqrt{\frac{\left(\frac{15}{\delta_{j^2}}\right)}{L}}\right) \\
&\leq 2R^2_{max}\left(\gamma^{T_J} + \sqrt{\frac{\left(\frac{15}{\delta_{j^2}}\right)}{L}}\right),
\end{aligned}$$

where the last inequality holds since $L \geq 1$ and $0 < \delta_{j^2} \leq 1$. □

Compared with the bound on $\hat{J}$ in Proposition 4.1, the main difference here is the larger constant $R^2_{\max}$, and that this bound grows faster when increasing confidence (i.e. decreasing δ) due to the use of Chebyshev's inequality. More importantly Proposition 4.2 shows that $\hat{J}^2$ is also a consistent estimator since we can achieve any level of accuracy with any level of confidence by sufficiently increasing L and T_J.

4.4 The Full Bound

We can now put everything together to display the full bound. First we state our assumption on the learned estimates for[8] V and M.

Assumption 4.1. *Let $\hat{V}$ and $\hat{M}$ be our learned estimates of V and M respectively, we assume that we are provided the following bounds with adjustable confidence parameters:*

- $\|V - \hat{V}\|_\mu \le \xi_v(\delta_v)$ *with probability* $1 - \delta_v$*, where* $0 < \delta_v \le 1$.
- $\|M - \hat{M}\|_\mu \le \xi_m(\delta_m)$ *with probability* $1 - \delta_m$*, where* $0 < \delta_m \le 1$.

Note that these bounds are written as functions of their confidence parameters, but they could depend on other quantities that we already using. For example, they could depend on T_J and L if $\hat{V}$ and $\hat{M}$ are learned from the same trajectories that were collected to learn $\hat{J}$. In its generality, the bound we provide in this section cannot be used to infer the sample complexity of the entire algorithm, unless ξ_v and ξ_m are provided in an explicit form that displays their dependence on the number of samples. We are now ready to state the main result of this chapter.

Theorem 4.1. *Suppose that Assumption 4.1 holds for our estimates $\hat{V}$ and $\hat{M}$, and that $\hat{J}$ is learned using Algorithm 3.1. If these estimates are used to learn $\hat{V}^\lambda$ using (4.1), we then have, with probability at least $1 - \delta$, that*

$$\|V^\lambda - \hat{V}^\lambda\|_\mu \le (1 + 2\lambda R_{\max})\xi_v\left(\frac{\delta}{4}\right) + \lambda\xi_m\left(\frac{\delta}{4}\right) + \frac{4\lambda R_{\max}^2}{1-\gamma}\left(\gamma^{T_J} + \sqrt{\frac{\left(\frac{60}{\delta}\right)}{L}}\right).$$

Proof. Starting from the bound in (4.3), we can apply Lemma 4.1 to get that

$$\begin{aligned}
&\|V^\lambda - \hat{V}^\lambda\|_\mu \\
&\le \|V - \hat{V}\|_\mu + 2\lambda R_{\max}\left(\frac{|J - \hat{J}|}{1-\gamma} + \|V - \hat{V}\|_\mu\right) + \lambda\|\hat{M} - M\|_\mu + \frac{\lambda}{1-\gamma}|\hat{J}^2 - J^2| \\
&= (1 + 2\lambda R_{\max})\|V - \hat{V}\|_\mu + \frac{2\lambda R_{\max}}{1-\gamma}|J - \hat{J}| + \lambda\|\hat{M} - M\|_\mu + \frac{\lambda}{1-\gamma}|\hat{J}^2 - J^2|.
\end{aligned}$$

We can then use Assumption 4.1, Proposition 4.1, and Proposition 4.2 (while setting $\delta_v = \delta_m = \delta_j = \delta_{j^2} = \frac{\delta}{4}$) to conclude that the following holds with probability at least $1 - \delta$:

$$\begin{aligned}
\|V^\lambda - \hat{V}^\lambda\|_\mu \le (1 + 2\lambda R_{\max})\xi_v\left(\frac{\delta}{4}\right) &+ \frac{2\lambda R_{\max}^2}{1-\gamma}\left(\gamma^{T_J} + \sqrt{\frac{2\log\left(\frac{8}{\delta}\right)}{L}}\right) \\
+ \lambda\xi_m\left(\frac{\delta}{4}\right) &+ \frac{2\lambda R_{\max}^2}{1-\gamma}\left(\gamma^{T_J} + \sqrt{\frac{\left(\frac{60}{\delta}\right)}{L}}\right)
\end{aligned}$$

[8]Remember that V is the state-value function, and M is the second moment value function defined in (3.8).

$$\leq (1+2\lambda R_{\max})\xi_v\left(\frac{\delta}{4}\right) + \lambda\xi_m\left(\frac{\delta}{4}\right) + \frac{4\lambda R_{\max}^2}{1-\gamma}\left(\gamma^{T_J} + \sqrt{\frac{\left(\frac{60}{\delta}\right)}{L}}\right),$$

where the last inequality holds since $2\log\left(\frac{8}{\delta}\right) \leq \frac{60}{\delta}$ for $0 < \delta \leq 1$. □

Next, we provide an example where we use least-squares temporal difference (introduced in Subsection 2.1.4) to learn $\hat{V}$ and $\hat{M}$, and we use the results in [38] to obtain explicit forms for ξ_v and ξ_m.

4.5 An Application of the Bound When Using LSTD

In [38], they provide finite sample analysis for a version of LSTD which they call *pathwise LSTD*. We will use this method to learn $\hat{V}$ and $\hat{M}$ using the same trajectory, and we will use Algorithm 3.1 to learn $\hat{J}$ independently. Although the bound of Theorem 4.1 does not assume that $\hat{V}$ and $\hat{M}$ are learned from different data, we will assume so since the bounds in [38] do not extend naturally to the case where we use multiple trajectories to learn the value function without having the bounds grow with the number of trajectories, which is problematic since the number of trajectories is a parameter that has to be increased to control the error on $\hat{J}$ and $\hat{J}^2$. Nonetheless, the algorithm we provide serves as a simple case where can exemplify the use of Theorem 4.1. Moreover, in [38], they provide two kinds of bounds on the accuracy of $\hat{V}$: an empirical bound and a generalization bound. In the empirical bound, the distribution of states according to which we evaluate $\hat{V}$ is the empirical distribution of the sampled trajectory. Whereas in the generalization bound, the stationary distribution of the Markov chain is used. In the following, we will use the empirical bound due to its simplicity. We can, in fact, use the generalization bound in exactly the same manner, but it contains a number of extra terms and requires a number of extra conditions that will distract from the main aim of the analysis.

Before describing our algorithm, we can provide some preliminaries adapted from [38]. Consider the basis functions $\varphi_i : \mathcal{S} \to \mathbb{R}, i = 1, \ldots, d$ defined over the states, and define $\phi(.) := (\varphi_1(.), ..., \varphi_d(.))^\intercal$ as the corresponding feature mapping. Furthermore, we assume that the basis functions are all uniformly bound by B (i.e. $|\varphi_i(s)| \leq B$ for $i = 1, \ldots, d$ and every $s \in \mathcal{S}$). Suppose we sample a trajectory of N steps $(s_1, a_1, r_1, \ldots, s_N, a_N, r_N)$, let $r \in \mathbb{R}^N$ and $r^2 \in \mathbb{R}^N$ be the vectors whose t$^{\text{th}}$ components are r_t and r_t^2 respectively. Moreover, for the same trajectory, define the features matrix $\Phi := [\phi(s_1); \ldots ; \phi(s_N)]$. For a vector $x \in \mathbb{R}^N$, define the following norm:

$$\|x\|_N^2 = \frac{1}{N}\sum_{t=1}^{N} x_t^2.$$

In the empirical bound of [38], they evaluate $\hat{V}$ according to $\|V - \hat{V}\|_N$. That is, we are only interested in the accuracy of $\hat{V}$ on the states of the trajectory each weighted

Algorithm 4.1 Factored Mean-Volatility pathwise LSTD

```
1: Input: a policy $\pi$ to be evaluated, feature Map $\phi(.)$, $\lambda$, $\gamma$, $N$, $L$, $T_J$.
2: $\hat{J}$ = Monte-Carlo-J$(\pi, \gamma, L, T_J)$. (see Algorithm 3.1.)
3: Use $\pi$ to sample a trajectory of $N$ steps $(s_1, a_1, r_1, \ldots, s_N, a_N, r_N)$.
4: Build the features matrix $\Phi = [\phi(s_1); \ldots ; \phi(s_N)]$.
5: Build the empirical transition matrix $\hat{P} : \hat{P}_{ij} = \mathbb{I}[j = i + 1, j \neq N]$.
6: Build matrix $A = \Phi^\intercal(I - \gamma\hat{P})\Phi$.
7: Build vector $b_v = \Phi^\intercal r$.
8: Build vector $b_m = \Phi^\intercal r^2$.
9: Set $\hat{\alpha}_v = A^+ b_v$. (This way, $\hat{V}(s) = \phi(s)^\intercal \hat{\alpha}_v \quad \forall s \in \mathcal{S}$.)
10: Set $\hat{\alpha}_m = A^+ b_m$. (This way, $\hat{M}(s) = \phi(s)^\intercal \hat{\alpha}_m \quad \forall s \in \mathcal{S}$.)
11: Set $\hat{V}^\lambda(s) = (1 + 2\lambda\hat{J})\phi(s)^\intercal\hat{\alpha}_v - \lambda\phi(s)^\intercal\hat{\alpha}_m - \frac{\lambda}{1-\gamma}\hat{J}^2 \quad \forall s \in \mathcal{S}$.
12: Output: $\hat{V}^\lambda$
```

by the frequency with which it was encountered. Thus, our function space is effectively the vector space $\mathcal{F}_N := \{\Phi\alpha : \alpha \in \mathbb{R}^d\}$, which is a subset of $\mathbb{R}^N$. Accordingly, define the operator $\hat{\Pi}$ which performs orthogonal projection to $\mathcal{F}_N$ according to the $\|.\|_N$ norm. That is, for a vector $y \in \mathbb{R}^N$, $\hat{\Pi}y = \arg\min_{z\in\mathbb{R}^N} \|y - z\|_N$. This way, $\hat{\Pi}V$ is the best function that we would hope to find in $\mathcal{F}_N$. Lastly, we note that the policy under evaluation is assumed to be deterministic in [38] for simplicity.

The pseudo-code of a factored mean-volatility policy evaluation algorithm that uses pathwise LSTD for learning $\hat{V}$ and $\hat{M}$ is provided in Algorithm 4.1. Note that A^+ denotes the Moore-Penrose pseudo-inverse of A, which is the same as A^{-1} whenever A is invertible [38]. The following theorem uses the bounds in [38] and Theorem 4.1 to derive an error bound on $\hat{V}^\lambda$ learned using Algorithm 4.1.

Theorem 4.2. *Suppose $\hat{V}^\lambda$ is obtained using Algorithm 4.1. Then, with probability at least $1 - \delta$, we have that*

$$\begin{aligned}\|V^\lambda - \hat{V}^\lambda\|_\mu &\leq \frac{1}{\sqrt{1-\gamma^2}}\Big((1 + 2\lambda R_{\max})\|V - \hat{\Pi}V\|_N + \lambda\|M - \hat{\Pi}M\|_N\Big) \\ &\leq (1 + 3\lambda R_{\max})\left[\frac{\gamma R_{\max}}{(1-\gamma)^2}B\sqrt{\frac{d}{v_N}}\left(\sqrt{\frac{8log(8d/\delta)}{N}} + \frac{1}{N}\right)\right] \\ &\quad + \frac{4\lambda R_{\max}^2}{1-\gamma}\left(\gamma^{T_J} + \sqrt{\frac{\left(\frac{60}{\delta}\right)}{L}}\right),\end{aligned}$$

where v_N is the smallest strictly positive eigenvalue of the matrix $\frac{1}{n}\Phi^\intercal\Phi$.

Proof. For $\hat{V}$, which we learned using pathwise LSTD, the bound in [38] states, with probability at least $1 - \delta_v$, that

$$\|V - \hat{V}\|_N \leq \frac{1}{\sqrt{1-\gamma^2}}\|V - \hat{\Pi}V\|_N + \frac{1}{1-\gamma}\left[\frac{\gamma R_{\max}}{1-\gamma}B\sqrt{\frac{d}{v_N}}\left(\sqrt{\frac{8log(2d/\delta_v)}{N}} + \frac{1}{N}\right)\right],$$

where v_N is the smallest strictly positive eigenvalue of the matrix $\frac{1}{n}\Phi^\intercal\Phi$. As for $\hat{M}$, as we remarked before, its analysis would be almost the same; we just have to replace $R_{\max}$ with $R^2_{\max}$. Thus, we have, with probability at least $1-\delta_m$, that

$$\|M-\hat{M}\|_N \leq \frac{1}{\sqrt{1-\gamma^2}}\|M-\hat{\Pi}M\|_N + \frac{1}{1-\gamma}\left[\frac{\gamma R^2_{\max}}{1-\gamma}B\sqrt{\frac{d}{v_N}}\left(\sqrt{\frac{8log(2d/\delta_m)}{N}}+\frac{1}{N}\right)\right].$$

We can then simply set $\delta_v = \delta_m = \frac{\delta}{4}$ and plug these bounds in the bound of Theorem 4.1 to get, with probability at least $1-\delta$, that

$$\begin{aligned}\|V^\lambda - \hat{V}^\lambda\|_\mu &\leq \frac{1}{\sqrt{1-\gamma^2}}\Big((1+2\lambda R_{\max})\|V-\hat{\Pi}V\|_N + \lambda\|M-\hat{\Pi}M\|_N\Big)\\ &\leq (1+3\lambda R_{\max})\left[\frac{\gamma R_{\max}}{(1-\gamma)^2}B\sqrt{\frac{d}{v_N}}\left(\sqrt{\frac{8log(8d/\delta)}{N}}+\frac{1}{N}\right)\right]\\ &\quad + \frac{4\lambda R^2_{\max}}{1-\gamma}\left(\gamma^{T_J} + \sqrt{\frac{\left(\frac{60}{\delta}\right)}{L}}\right),\end{aligned}$$

which is the desired statement. □

The first term in the bound is the irreducible approximation error caused by the limited representation power of the linear space we are using. The second term is the estimation error on $\hat{V}$ and $\hat{M}$, which can be reduced by increasing the length of the trajectory used by LSTD. Finally, the last term is the error due to the inaccuracy of $\hat{J}$ and $\hat{J}^2$, which can be reduced by increasing L and T_J in Algorithm 3.1. The bound in Theorem 4.2 implies that the sample complexity of the algorithm, for achieving $\|V^\lambda - \hat{V}^\lambda\|_\mu \leq \epsilon$ for a sufficiently small ϵ with a fixed level of confidence, is $LT_J + N = \mathcal{O}(\epsilon^{-2}\log(\epsilon^{-1}) + \epsilon^{-2}) = \mathcal{O}(\epsilon^{-2}\log(\epsilon^{-1}))$. This is compared to the $\mathcal{O}(\epsilon^{-2})$ sample complexity for learning only the risk-neutral value function V using pathwise LSTD. It would be interesting to see if the complexity of Algorithm 4.1 can be improved if $\hat{V}$ and $\hat{M}$ are learned using the same trajectories collected for estimating the expected return, but as remarked before, the analysis in [38] does not seem to be naturally extendable to the case where we use multiple trajectories without causing the bound to grow with the number of trajectories.[9]

[9]Note that, with the same tools, we can propose a small variant of Algorithm 4.1 in which the trajectory used to learn $\hat{V}$ and $\hat{M}$ is *one* of the trajectories used to learn $\hat{J}$. This way, we would replace N with T_J. This might seems like a sound idea since we are reusing samples. However, this will have a negative effect on sample complexity since now the complexity will be given by $LT_J = \mathcal{O}(\epsilon^{-2}\epsilon^{-2}) = \mathcal{O}(\epsilon^{-4})$ as this scheme causes the bound to contain a term of order $\mathcal{O}(\sqrt{\frac{1}{T_J}})$, which dominates the γ^{T_J} term. This negative effect is not surprising since we are forcing all the trajectories to be as long as the one we are using for LSTD. Alternatively, one can adhere to the sampling scheme of the original algorithm, but also use the LSTD trajectory as one of the trajectories used to estimate J. However, this does not seem to bring any significant advantage in terms of sample efficiency.

Chapter 5

Analyzing a Mean-Volatility Actor-Critic Algorithm

In this chapter, we provide finite sample analysis of a complete actor-critic algorithm that aims to optimize the mean-volatility objective. More specifically, we adapt the (risk-neutral) mini-batch actor-critic algorithm analyzed in [11] to our setting by firstly estimating the expected return, and then using it to transform the rewards that are used by the algorithm. This way, at each iteration of the actor-critic algorithm, we perform policy evaluation using the mini-batch semi-gradient TD algorithm (described in Subsection 2.1.3) while adapting it to the mean-volatility objective using the direct method (See Algorithm 3.3). We will start by describing in more detail the algorithm used in [11] and how we adapt it to our risk-averse setting. After presenting some preliminaries, we proceed with the finite sample analysis of the critic, and finally conclude with the analysis of the actor.

Throughout this chapter, we still adopt the MDP $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \mu_0 \rangle$ that we have used in the last section. Moreover, we assume, like in Section 2.2, that the policies we consider are from a class Π of policies parameterized by a vector $\theta \in \mathbb{R}^{d_\theta}$, and $\pi_\theta(a|s)$ is continuously differentiable with respect to θ for any state-action pair. Our aim is still to find a policy π_θ maximizing[1] $\eta_\theta := J_\theta - \lambda \nu_\theta^2$, where J_θ is the expected return (defined in (3.1)), ν_θ^2 is the reward volatility (defined in (3.4)), and $\lambda \geq 0$ is a parameter used to control the trade-off between maximizing J_θ and minimizing ν_θ^2. Remember also the expression (or one form of it) for the gradient of the mean volatility with respect to the parameters of the policy [7]:

$$\nabla_\theta \eta_\theta = \mathop{\mathbb{E}}_{\substack{s \sim d_{\mu_0, \pi_\theta}(\cdot) \\ a \sim \pi_\theta(\cdot|s)}} \Big[A^\lambda_{\pi_\theta}(s,a) \nabla_\theta \log \pi_\theta(a|s) \Big], \tag{5.1}$$

where $A^\lambda_\pi(s,a) := Q^\lambda_\pi(s,a) - V^\lambda_\pi(s)$ is the transformed advantage function that we introduced before, whereas V^λ_π and Q^λ_π are, respectively, the transformed state-value and action-value functions introduced in Section 3.1.

[1]Remember the convention of writing θ instead of π_θ when the context is clear.

Remember from our discussion in Section 2.2 that policy gradient methods aim to optimize the policy by generating a sequence of parameter vectors $\{\theta_t\}$ each of which is obtained from the previous by moving along the gradient of the objective function, that is

$$\theta_{t+1} = \theta_t + \alpha_t \nabla \eta_{\theta_t}, \tag{5.2}$$

where α_t is the step size at step t. More specifically, we focus on actor-critic methods, which are policy gradient methods that use a learned estimate of the value function that is to be used in the policy gradient expression (e.g. A_π^λ in (5.1)) at each iteration of the algorithm. In the approach we consider in this chapter, we do not learn an estimate of A_π^λ directly. Rather, we learn an estimate of V_π^λ and use it to form estimates of A_π^λ at the required state-action pairs (see (2.17) or (5.4) below for an example of this approach). To this end, we consider a linear approximation scheme for V_π^λ where candidate functions belong to the function space $\{V_\omega^\lambda : \omega \in \mathbb{R}^{d_\omega}$ and $V_\omega^\lambda(.) = \omega^\intercal \phi(.)\}$, where $\varphi_i : S \to \mathbb{R}, i = 1, \ldots, d_\omega$. are basis functions defined over the states, and $\phi(.) := (\varphi_1(.), \ldots, \varphi_{d_\omega}(.))^\intercal$ is the corresponding feature mapping.

5.1 The Algorithm

In [11], they provide finite sample analysis for a (risk-neutral) actor-critic algorithm that uses a linear TD(0) critic (which we investigated in some detail in 2.1.2). More specifically, they use a mini-batch version of linear TD(0) (briefly discussed in Subsection 2.1.3) in which a mini-batch of samples is used to performs the updates: (recall from (2.7))

$$\omega_{i+1} := \omega_i + \alpha \frac{1}{M} \sum_{t=0}^{M-1} [R_{i,t} + \gamma \phi(s_{i,t+1})^\intercal \omega_i - \phi(s_{i,t})^\intercal \omega_i] \phi(s_{i,t}),$$

this is opposed to using just a single sample (as in (2.2)). Their motivation for adopting mini-batch updates is that the iterates can be driven arbitrarily close, in expectation, to the TD fixed point by increasing the mini-batch size while using a fixed step-size. Using this approach, they were able prove a better sample complexity than what was available in the literature at the time (e.g. [12]). As for the actor, they adopt an A2C approach while also using mini-batches to perform the updates. That is, they use a mini-batch version of (2.17):

$$\theta_{t+1} = \theta_t + \alpha \frac{1}{B} \sum_{i=0}^{B-1} (R(s_{t,i}, a_{t,i}) + \gamma \phi(s_{t,i+1})^\intercal \omega_t - \phi(s_{t,i})^\intercal \omega_t) \nabla_\theta \log \pi_{\theta_t}(a_{t,i}|s_{t,i}). \tag{5.3}$$

Recall from our discussion in Section 2.2 that, in practice, policy gradient methods use a stochastic estimate of the gradient, which we obtain using samples collected while interacting with the environment. In particular, we discussed two approaches to force the sampled states to obey the discounted state distribution $d_{\mu_0,\pi_\theta}(\cdot)$, which we recall its definition from (2.12):

$$d_{\mu_0,\pi_\theta}(s) := (1-\gamma) \mathop{\mathbb{E}}_{s_0 \sim \mu_0(\cdot)} \left[\sum_{t=0}^{\infty} \gamma^t p_{\pi_\theta}(s_0 \xrightarrow{t} s) \right].$$

The first approach is to interact with the environment normally (yielding samples drawn from the on-policy distribution) while discounting the resulting gradient estimates depending on the time-step at which each sample was encountered (see (2.13) for an example). The other approach, which is adopted in [11], is to interact with a slightly modified MDP characterized by the following transition kernel:

$$\tilde{P}(\cdot|s,a) = \gamma P(\cdot|s,a) + (1-\gamma)\mu_0(\cdot),$$

where P is the transition kernel of the original MDP. That is, at each step, we sample the next state according to the original kernel with probability γ, while we draw the next state from the initial state distribution with probability $1-\gamma$. In [11], at each iteration, after having trained the critic, they interact with the modified kernel to obtain a mini-batch of samples to be used to estimate the gradient. However, directly adopting this approach leads to a subtle bias in their algorithm (and analysis). To see this, recall from (2.17) the (single sample) A2C update:

$$\theta_{t+1} := \theta_t + \alpha_t \gamma^t (R_t + \gamma V_{\omega_t}(s_{t+1}) - V_{\omega_t}(s_t)) \nabla_\theta \log \pi_{\theta_t}(a_t|s_t). \tag{5.4}$$

Here, $R_t + \gamma V_{\omega_t}(s_{t+1}) - V_{\omega_t}(s_t)$ is effectively a stochastic estimate of the advantage function at (s_t, a_t), where $V_{\omega_t}(s_t)$ is our estimate (the critic) of the state-value function at s_t, while $R_t + \gamma V_{\omega_t}(s_{t+1})$ is an estimate of the action-value function at (s_t, a_t). Recalling that $Q^\pi(s_t, a_t) = R(s_t, a_t) + \gamma\, \mathbb{E}_{s_{t+1}\sim P(\cdot|s_t,a_t)}[V^\pi(s_{t+1})]$, to yield unbiased estimates, the next state (s_{t+1}) has to be properly sampled from $P(\cdot|s_t, a_t)$. However, in [11], they (erroneously)[2] use the sampled next state in the actor mini-batch, which is actually sampled from the MDP with the modified kernel $\tilde{P}$. To remedy this, we employ an alternative sampling process, which allows us to both (1) sample s_{t+1} (like in (5.4)) according to the original kernel, and (2) obtain an unbiased estimate of the gradient. Consider two different random variables for the state, namely, s_{t+1} and s'_{t+1}, with different distributions. The latter one is distributed according to the standard kernel, $s'_{t+1} \sim P(\cdot|s_t, a_t)$, while s_{t+1} is sampled from the following variant of the modified kernel $\tilde{P}(\cdot|s_t, a_t, s'_{t+1}) := \gamma\delta_{s'_{t+1}}(\cdot) + (1-\gamma)\mu_0(\cdot)$. That is, with probability γ, s_{t+1} is the same as s'_{t+1}, and with probability $1-\gamma$, s_{t+1} is drawn from the initial state distribution. In any case, s'_{t+1} is the one we use as the next state in the gradient estimate (5.3), whereas s_{t+1} is the state from which we resume sampling the rest of the actor mini-batch.

Note that the proposed sampling approach does not require a *generative* model (i.e. one in which we can, at any time, sample (multiple) transitions from any state we want), it only requires that we can halt the trajectory at any time and restart from the initial state distribution. With the proposed modification, the analysis of [11] is largely unaffected, we just need to account for the extra performed sampling when we consider the sample complexity of the algorithm. For now, we will present the algorithm for which we will extend the analysis in [11]. Aside from adopting the proposed new sampling scheme, our algorithm differs from the one considered in [11] in that, at iteration, it learns an estimate of the expected return of the current policy (using Algorithm 3.1) and uses it

[2]Note that the estimates of the advantage function are already biased due to the use of the (usually) inaccurate estimate of the state-value function. However, this bias, unlike the one caused by sampling from the wrong kernel, is actually accounted for in the analysis.

Algorithm 5.1 Mini-batch Mean-Volatility Actor-Critic (Mini-batch MVAC)

```
1: Input: Policy Class $\pi_\theta$, Feature Map $\phi(.)$, $\mu_0(.), \alpha, \beta, \lambda, \gamma, L, T_J, T_c, M, T, B$
2: Initialize: $\theta_0$
3: for $t = 0, \dots, T-1$ do
4:     if $t = 0$ then
5:         $s_{\text{ini}} \sim \mu_0(.)$
6:     else
7:         $s_{\text{ini}} = s_{t-1,B}$
8:     end if
9:     Estimating J: $\hat{J}_t$ = Monte-Carlo-J($\pi_{\theta_t}, \gamma, L, T_J$) (see Algorithm 3.1.)
10:    Critic update: $\omega_t, s_{t,0}$ = Mini-batch-MVTD($s_{\text{ini}}, \theta_t, \phi, \gamma, \beta, T_c, M, \lambda, \hat{J}_t$)
   (see Algorithm 5.2.)
11:    Actor mini-batch sampling:
12:    for $i = 0, \dots, B-1$ do
13:        $a_{t,i} \sim \pi_\theta(s_{t,i})$
14:        $s'_{t,i+1} \sim P(\cdot|s_{t,i}, a_{t,i})$
15:        $s_{t,i+1} \sim \tilde{P}(\cdot|s_{t,i}, a_{t,i}, s'_{t,i+1})$
16:        $\delta_{\omega_t}(s_{t,i}, a_{t,i}, s'_{t,i+1}) = R^\lambda(s_{t,i}, a_{t,i}, \hat{J}_t) + \gamma\phi(s'_{t,i+1})^\intercal\omega_t - \phi(s_{t,i})^\intercal\omega_t$
17:    end for
18:    Actor update: $\theta_{t+1} = \theta_t + \alpha\frac{1}{B}\sum_{i=0}^{B-1}\delta_{\omega_t}(s_{t,i}, a_{t,i}, s'_{t,i+1})\psi_{\theta_t}(s_{t,i}, a_{t,i})$
19: end for
20: Output: $\theta_{\hat{T}}$ with $\hat{T}$ chosen uniformly from $\{1, \dots, T\}$.
```

to transform the rewards used for training the critic (i.e. we use the direct method) and the rewards used to form the estimates of the advantage function used in the actor update. Before proceeding, we define following two functions:

$$R^\lambda(s, a, \hat{J}) := \left[R(s,a) - \lambda(R(s,a) - \hat{J})^2\right],$$

and

$$\psi_\theta(s,a) := \nabla \log \pi_\theta(a|s).$$

We refer to the latter as the *score function* of policy π_θ. The pseudo-code of the full algorithm is presented in Algorithm 5.1, and the critic sub-routine it uses is presented in Algorithm 5.2.

To clarify the used parameters: T is the number of iterations of the full algorithm (i.e. number of times we update the policy using gradient ascent), B is the size of the actor's mini-batch, T_c is the number of critic iterations (i.e. the number of semi-gradient TD updates), M is the size of the critic's mini-batch, and finally L and T_J are the usual number of trajectories and horizon length used in Algorithm 3.1. Note that, at each iteration, the proposed algorithm uses different samples for forming each of the expected return estimate, the value function estimate, and the gradient estimate. In particular, as in [11], the algorithm uses (aside from the trajectories collected by Algorithm 3.1)

Algorithm 5.2 Mini-batch Mean-Volatility TD (Mini-batch MVTD)

```
1: Input: $s_{\text{ini}}, \theta, \phi, \gamma, \beta, T_c, M, \lambda, \hat{J}$
2: Initialize: $\omega_0$
3: for $k = 0, \dots, T_c - 1$ do
4:     if $k = 0$ then
5:         $s_{k,0} = s_{\text{ini}}$
6:     else
7:         $s_{k,0} = s_{k-1,M}$
8:     end if
9:     for $j = 0, \dots, M - 1$ do
10:        $a_{k,j} \sim \pi_\theta(s_{k,j}),\ s_{k,j+1} \sim P(.|s_{k,j}, a_{k,j})$
11:        $\delta_{\omega_k}(s_{k,j}, a_{k,j}, s_{k,j+1}) = R^\lambda(s_{k,j}, a_{k,j}, \hat{J}) + \gamma\phi(s_{k,j+1})^\intercal\omega_k - \phi(s_{k,j})^\intercal\omega_k$
12:    end for
13:    $\omega_{k+1} = \omega_k + \beta\frac{1}{M}\sum_{j=0}^{M-1}\delta_{\omega_k}(s_{k,j}, a_{k,j}, s_{k,j+1})\phi(s_{k,j})$
14: end for
15: Output: $\omega_{T_c}, s_{k,M}$
```

a "single" sample path in the sense that the actor starts sampling from the last state of the critic mini-batch, and vice versa. However, calling it a single sample path is not very accurate since the actor samples from the modified kernel, which introduces the possibility, at any step, of restarting the trajectory. Finally, we note that the motivation for the output policy parameter vector to be randomly picked uniformly across all the iterations is purely theoretical since, as we will discuss, our goal when analyzing the algorithm is to show that, on average (across the iterations), the norm of the gradient of the mean-volatility vanishes (i.e. we approach a stationary point) as the number of used samples increases. In practice, one can estimate the performance of the obtained policy at each iteration[3], and output at the end the best encountered policy.

5.2 Preliminaries

We can start by making a number of technical assumptions (some are adapted from [11]).

Assumption 5.1. $\forall (s, a) \in S \times A$

- **(i)** $|R(s, a)| \le R_{max}$.
- **(ii)** *$\pi_\theta(a|s)$ is differentiable w.r.t. θ.*
- **(iii)** $\exists\, C_\psi > 0 : \forall \theta\ \|\psi_\theta(s, a)\|_2 \le C_\psi$.
- **(iv)** $\exists\, L_\psi > 0 : \forall \theta_1, \theta_2\ \|\psi_{\theta_1}(s, a) - \psi_{\theta_2}(s, a)\|_2 \le L_\psi\|\theta_1 - \theta_2\|_2$.

[3]Which can be done, for example, by taking the trajectories already collected by Algorithm 3.1, transforming their rewards using $\hat{J}$, and using them to obtain a Monte-Carlo estimate of the mean-volatility.

(v) $\exists\, C_\pi > 0 : \forall \theta_1, \theta_2\ \|\pi_{\theta_1}(.|s) - \pi_{\theta_2}(.|s)\|_{TV} \leq C_\pi \|\theta_1 - \theta_2\|_2$,

where, for a probability density function q(.), $\|q(.)\|_{TV} := \frac{1}{2}\int_s |q(ds)|$.

Assumptions 5.1.iii and 5.1.iv assert that, for any policy in our class of policies, the score function is bounded and smooth, while assumption 5.1.v asserts that the chosen class of policies is smooth in the described sense. Note that by Assumption 5.1.i and the definition of J_π in (3.1), $\forall (s,a) \in S \times A$ and $\lambda \geq 0$,

$$\left|R(s,a) - \lambda(R(s,a) - J_\pi)^2\right| \leq R_{\lambda,\max} := R_{\max} + 4\lambda R_{\max}^2.$$

In the following, we may refer to $\left[R(s,a) - \lambda(R(s,a) - J_\pi)^2\right]$ as $R_\pi^\lambda(s,a)$. We also make the following assumption on the basis functions and the feature mapping that we use to learn V_π^λ.

Assumption 5.2. $\exists\, C_\phi > 0 : \forall s \in S\ \|\phi(s)\|_2 \leq C_\phi$. *Furthermore, the basis functions* $\varphi_i(\cdot), i = 1, ..., d_\omega$ *are mutually linearly independent.*

The following assumption serves to simplify the expressions of the bounds.

Assumption 5.3. *W.L.O.G.*

(i) $C_\psi = 1$.

(ii) $C_\phi = 1$.

The following is an assumption on the regularity of the MDP.

Assumption 5.4 (Uniform Ergodicity, Adapted from [11])**.** *For any* $\theta \in \mathbb{R}^{d_\theta}$, *consider the MDP with policy* π_θ *and the transition kernel* $P(\cdot|s,a)$ *or* $\tilde{P}(\cdot|s,a) = \gamma P(\cdot|s,a) + (1-\gamma)\xi(\cdot)$, *where* $\xi(\cdot)$ *can be* μ_0 *or* $P(\cdot|\hat{s},\hat{a})$ *for any* $(\hat{s},\hat{a}) \in \mathcal{S} \times \mathcal{A}$. *Let* μ_{π_θ} *be the stationary state distribution of the MDP when acting with policy* π_θ. *There exists constants* $\kappa > 0$ *and* $\rho \in (0,1)$ *such that*

$$\sup_{s \in \mathcal{S}} \|\mathbb{P}(s_t \in \cdot | s_0 = s) - \mu_{\pi_\theta}(\cdot)\|_{TV} \leq \kappa \rho^t, \quad \forall t \geq 0.$$

In Subsection 2.1.2, we discussed the semi-gradient TD(0) algorithm with linear function approximation. In particular, we discussed its expected behaviour in steady state, and characterized the point to which it converges. Since our critic is a mean-volatility version of this algorithm (i.e. with transformed rewards $R^\lambda(s,a,\hat{J})$, where $\hat{J}$ is our learned estimate of the expected return of the policy under evaluation), we can define similar quantities to those defined in the risk-neutral case. Given a policy π_θ with stationary state distribution μ_θ, consider the update rule of the critic parameters at the i^{th} iteration in the critic sub-routine:

$$\begin{aligned}
\omega_{i+1} &= \omega_i + \beta \frac{1}{M} \sum_{t=0}^{M-1} \delta_{\omega_i}(s_{i,t}, a_{i,t}, s_{i,t+1}) \phi(s_{i,t}) \\
&= \omega_i + \beta \frac{1}{M} \sum_{t=0}^{M-1} \Big[R^\lambda(s_{i,t}, a_{i,t}, \hat{J}) + \gamma \phi(s_{i,t+1})^\intercal \omega_i - \phi(s_{i,t})^\intercal \omega_i \Big] \phi(s_{i,t}) \\
&= \omega_i + \beta \left(\frac{1}{M} \sum_{t=0}^{M-1} \Big[\phi(s_{i,t}) R^\lambda(s_{i,t}, a_{i,t}, \hat{J}) \Big] + \frac{1}{M} \sum_{t=0}^{M-1} [\phi(s_{i,t})(\gamma \phi(s_{i,t+1}) - \phi(s_{i,t}))^\intercal] \, \omega_i \right)
\end{aligned}$$

Define $b_{i,t}(\hat{J}) := \phi(s_{i,t}) R^\lambda(s_{i,t}, a_{i,t}, \hat{J})$, and $A_{i,t} := \phi(s_{i,t})(\gamma \phi(s_{i,t+1}) - \phi(s_{i,t}))^\intercal$. Correspondingly, define $\hat{b}_i(\hat{J}) := \frac{1}{M} \sum_{t=0}^{M-1} b_{i,t}(\hat{J})$, and $\hat{A}_i := \frac{1}{M} \sum_{t=0}^{M-1} A_{i,t}$. Also, define the shorthand function $g_i(\omega, \hat{J}) = \hat{b}_i(\hat{J}) + \hat{A}_i \, \omega$. We can thus rewrite the critic update as

$$\begin{aligned}
\omega_{i+1} &= \omega_i + \beta \Big(\hat{b}_i(\hat{J}) + \hat{A}_i \, \omega_i \Big) \\
&= \omega_i + \beta g_i(\omega_i, \hat{J}).
\end{aligned}$$

Similar to what we did in the risk-neutral case, we can describe a deterministic version of the algorithm where we use the expected value of $\hat{b}$ and $\hat{A}$ when the MDP has reached steady-state. To this end, we can define $b(\hat{J}) := \mathbb{E}_{\mu_\theta} \Big[\phi(s_t) R^\lambda(s_t, a_t, \hat{J}) \Big]$, $A := \mathbb{E}_{\mu_\theta}[\phi(s_t)(\gamma \phi(s_{t+1}) - \phi(s_t))^\intercal]$, and $\bar{g}(\omega, \hat{J}) := b(\hat{J}) + A \, \omega$. We can then write the deterministic update rule as

$$\omega_{i+1} = \omega_i + \beta \Big(b(\hat{J}) + A \, \omega_i \Big). \tag{5.5}$$

For a fixed $\hat{J}$, the policy evaluation problem at hand is not at all different from the risk-neutral version, albeit with a modified reward function. This means that the results in [9] about temporal difference learning with linear function approximation are applicable in this case. Most notably, it means that the matrix A (which is independent of the reward function) is negative definite, and the (unique) stationary point $\omega^*_{\hat{J}}$ of (5.5) (i.e. $A\omega^*_{\hat{J}} + b(\hat{J}) = 0$) is the fixed point of the projected Bellman equation (see (2.6)) on the transformed value function. What we are actually interested in is describing the convergence rate (in expectation)[4] of the critic to ω^*_J, where J is the true (normalized) expected return of the policy under evaluation. More specifically, the goal is to bound $\mathbb{E}\big[\|\omega_{T_c} - \omega^*_J\|_2^2\big]$ in terms of the number of used samples, where T_c is the number of iterations of the critic. If we were provided with the real value of J, one could then use it to transform the rewards, and in that case, we can directly inherit the results in [11]. However, we do not have access to such value, and our only option is to use our

[4]The reason why we are deriving a bound in expectation is for the bound to be usable in the analysis of the actor, as we will see later.

Monte-Carlo estimate $\hat{J}$ in its place[5]. And therein lies the challenge of analyzing the critic: understanding the effect of using $\hat{J}$ instead of J in the reward transformation. Before we proceed, we just have to mention a few more technical points, starting with the following assumption (which can be justified since we assumed, in Assumptions 5.1 and 5.2, that the feature mapping and the reward function are bounded).

Assumption 5.5. *For any triple $(s_{i,t}, a_{i,t}, s_{i,t+1}) \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ and any $\hat{J}$ estimate bounded, in absolute value, by $R_{\max}$, there exists real constants C_A and C_b such that $\|A_{i,t}\|_F \leq C_A$ and $\|b_{i,t}(\hat{J})\|_2 \leq C_b$, where $\|\cdot\|_F$ is the Frobenius norm[6] of a matrix.*

In [11], they state the following as a result[7] of using bounded and independent features: there exists a positive constant λ_A such that, for any $\omega \in \mathbb{R}^{d_\omega}$, we have that

$$\langle(\omega - \omega^*), A(\omega - \omega^*)\rangle \leq -\frac{\lambda_A}{2}\|\omega - \omega^*\|_2^2,$$

where ω^* is the TD fixed point in the risk-neutral case. The stated inequality, in fact, can be seen as general property of matrix A independently of ω^* since any vector $x \in \mathbb{R}^{d_\omega}$ can be written as $(x+\omega^*)-\omega^*$. We can the restate this inequality in a more convenient form for us: there exists a positive constant λ_A such that, for any $\omega \in \mathbb{R}^{d_\omega}$ and any value of our (bounded) estimate $\hat{J}$, we have that

$$\langle(\omega - \omega^*_{\hat{J}}), A(\omega - \omega^*_{\hat{J}})\rangle \leq -\frac{\lambda_A}{2}\left\|\omega - \omega^*_{\hat{J}}\right\|_2^2.$$

Our assumptions should also guarantee that, for any $\hat{J} \leq R_{max}$, $\|\omega^*_{\hat{J}}\|_2$ is uniformly upper bounded by some positive constant C_ω. To see this, first note that $\|\omega^*_{\hat{J}}\|_2 = \|A^{-1}b(\hat{J})\|_2$. For an $n \times m$ matrix A, its spectral norm (or induced l2 norm) is defined as $\|A\|_2 = \sup_{x\neq 0}\frac{\|Ax\|_2}{\|x\|_2}$. This means that $\|Ax\|_2 \leq \|A\|_2\|x\|_2$, for any vector $x \in \mathbb{R}^m$. For us, this means that $\|A^{-1}b(\hat{J})\|_2 \leq \|A^{-1}\|_2\|b(\hat{J})\|_2 \leq \frac{C_b}{\bar{\sigma}} := C_\omega$, where we denote by $\bar{\sigma}$ the smallest singular value of A, and the last inequality follows by Assumption 5.5 and Lemma A.1.

As for the actor analysis, we are ideally interested in finding an optimal policy (i.e. a parameter vector θ that maximizes[8] $\eta(\theta)$). However, since $\eta(\theta)$ is, in general, a non-concave function of θ, we do not expect, in general, that we reach a global maximum using a gradient ascent algorithm. Instead of a global optimum, we strive to reach a stationary point of $\eta(\theta)$, and the goal of the analysis is thus to bound $\mathbb{E}\left[\left\|\nabla\eta(\theta_{\hat{T}})\right\|_2^2\right]$ in terms of the number of used samples. Crucial to the analysis of the actor is for the gradient of $\eta(\theta)$ to be Lipschitz continuous. That is, for any $\theta_1, \theta_2 \in \mathbb{R}^{d_\theta}$, there exists a

[5]Note that $\hat{J}$ is not a fixed value, it is a random variable whose randomness comes from the sampling process that is used to learn it.

[6]For an $m \times n$ matrix X, its Frobenius norm [43] is defined as $\|X\|_F := \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2} = \sqrt{\sum_{i=1}^{\min\{m,n\}} \sigma_i^2(A)}$, where $\sigma_i(A)$ are the singular value of A.

[7]This can be seen as a consequence of Lemmas 1 and 3 in [12].

[8]$\eta(\theta) := \eta_\theta$.

real constant $L_\eta \geq 0$ such that

$$\|\nabla\eta(\theta_1) - \nabla\eta(\theta_2)\|_2 \leq L_\eta\|\theta_1 - \theta_2\|_2.$$

In [11], they (instead of assuming it) prove that this property holds for J_θ given the assumptions made so far. In Appendix B, we mirror their analysis for our case and provide proof of the stated property for η, along with other intermediary results. Before proceeding with the analysis of the critic and actor parts, we first need to adapt the results from Section 4.3 to obtain bounds *in expectation* on $\hat{J}$ and $\hat{J}^2$.

5.3 Analyzing the Monte-Carlo Estimation of the Expected Return

In Section 4.3, we derived probability bounds on $|J - \hat{J}|$ and $|J^2 - \hat{J}^2|$, where J is the (normalized) expected return of the policy under evaluation and $\hat{J}$ is the Monte-Carlo estimate of J obtained using Algorithm 3.1. Unfortunately, these bounds are not directly usable in the analysis to carried out in this chapter since we are deriving bounds in expectation not in probability. In this section, we adapt the results from Section 4.3 to bound the two quantities $\mathbb{E}\left[\left(J - \hat{J}\right)^2\right]$ and $\mathbb{E}\left[\left(J^2 - \hat{J}^2\right)^2\right]$. The following proposition provides the first bound. *(Note that the proofs in this section rely on quantities defined in Section 4.3, please refer to the summary at the beginning of that section.)*

Proposition 5.1. *Suppose, for a given policy, an estimate $\hat{J}$ is obtained using Algorithm 3.1, and that Assumption 5.1.i holds, then we have*

$$\mathbb{E}\left[\left(J - \hat{J}\right)^2\right] \leq \gamma^{2T_J} R_{\max}^2 + \frac{R_{\max}^2}{L}.$$

Proof. We begin with a bias-variance decomposition:

$$\begin{aligned}\mathbb{E}\left[\left(J - \hat{J}\right)^2\right] &= \mathbb{E}\left[\left(J - \mathbb{E}[\hat{J}] + \mathbb{E}[\hat{J}] - \hat{J}\right)^2\right] \\ &= \mathbb{E}\left[(J - \mathbb{E}[\hat{J}])^2\right] + \mathbb{E}\left[\left(\mathbb{E}[\hat{J}] - \hat{J}\right)^2\right] \\ &= \left(J - \mathbb{E}[\hat{J}]\right)^2 + Var(\hat{J}),\end{aligned}$$

where the second equality holds since $2\left(J - \mathbb{E}[\hat{J}]\right)\mathbb{E}\left[\left(\mathbb{E}[\hat{J}] - \hat{J}\right)\right] = 0$. For the bias term, we know from Lemma 4.3 that

$$\left|J - \mathbb{E}[\hat{J}]\right| \leq \gamma^{T_J} R_{\max}.$$

Thus, $\left(J - \mathbb{E}[\hat{J}]\right)^2 \leq \gamma^{2T_J} R_{\max}^2$. As for the variance, since $\hat{J}$ is a sample mean of $G_0, ..., G_{L-1}$, then $Var(\hat{J}) = \frac{\zeta_2}{L}$. Combining both terms and applying Lemma 4.2, we get

$$\mathbb{E}\left[\left(J-\hat{J}\right)^2\right] \leq \gamma^{2T_J} R_{\max}^2 + \frac{\zeta_2}{L} \leq \gamma^{2T_J} R_{\max}^2 + \frac{R_{\max}^2}{L}.$$

□

The next proposition provides a bound on the expected squared difference between $\hat{J}^2$ and J^2.

Proposition 5.2. *Suppose, for a given policy, an estimate $\hat{J}$ is obtained using Algorithm 3.1, and that Assumption 5.1.i holds, then we have*

$$\begin{aligned}\mathbb{E}\left[\left(J^2-\hat{J}^2\right)^2\right] &\leq 4R_{\max}^4\gamma^{2T_J} + 4R_{\max}^2\frac{\zeta_2}{L} + \frac{3\zeta_2^2 + 4|\zeta_3|R_{\max}}{L^2} + \frac{\zeta_4 - 3\zeta_2^2}{L^3} \\ &\leq 4R_{\max}^4\gamma^{2T_J} + R_{\max}^4\left(\frac{4}{L} + \frac{7}{L^2} + \frac{5}{L^3}\right).\end{aligned}$$

Proof. We, again, start with a bias-variance decomposition:

$$\begin{aligned}\mathbb{E}\left[\left(J^2-\hat{J}^2\right)^2\right] &= \mathbb{E}\left[\left(J^2 - \mathbb{E}\left[\hat{J}^2\right] + \mathbb{E}\left[\hat{J}^2\right] - \hat{J}^2\right)^2\right] \\ &= \mathbb{E}\left[\left(J^2 - \mathbb{E}\left[\hat{J}^2\right]\right)^2\right] + \mathbb{E}\left[\left(\mathbb{E}\left[\hat{J}^2\right] - \hat{J}^2\right)^2\right] \\ &= \left(J^2 - \mathbb{E}\left[\hat{J}^2\right]\right)^2 + Var(\hat{J}^2).\end{aligned}$$

For the bias term, similar to what we did in Lemma 4.4, we have that

$$\begin{aligned}\left|J^2 - \mathbb{E}[\hat{J}^2]\right| &= \left|J^2 - \mathbb{E}[\hat{J}]^2 - \frac{\zeta_2}{L}\right| \\ &\leq \left|J^2 - \mathbb{E}[\hat{J}]^2\right| + \frac{\zeta_2}{L} \\ &= \left|(\overline{G}_{0:T_J-1} + \overline{G}_{T_J:\infty})^2 - \overline{G}_{0:T_J-1}^2\right| + \frac{\zeta_2}{L} \\ &= \left|\overline{G}_{T_J:\infty}(2\overline{G}_{0:T_J-1} + \overline{G}_{T_J:\infty})\right| + \frac{\zeta_2}{L} \\ &\leq \gamma^{T_J}R_{\max}\left(2(1-\gamma^{T_J})R_{\max} + \gamma^{T_J}R_{\max}\right) + \frac{\zeta_2}{L} \\ &= \gamma^{T_J}(2-\gamma^{T_J})R_{\max}^2 + \frac{\zeta_2}{L} \\ &\leq 2\gamma^{T_J}R_{\max}^2 + \frac{\zeta_2}{L}.\end{aligned}$$

Thus,

$$\left(J^2 - \mathbb{E}\left[\hat{J}^2\right]\right)^2 \leq \left(2\gamma^{T_J}R_{\max}^2 + \frac{\zeta_2}{L}\right)^2 = 4\gamma^{2T_J}R_{\max}^4 + 4\gamma^{T_J}R_{\max}^2\frac{\zeta_2}{L} + \frac{\zeta_2^2}{L^2}.$$

For the variance, we apply Lemma 4.5:

$$Var(\hat{J}^2) = 4\overline{G}_{0:T_J-1}^2 \frac{\zeta_2}{L} + \frac{2\zeta_2^2 + 4\zeta_3\overline{G}_{0:T_J-1}}{L^2} + \frac{\zeta_4 - 3\zeta_2^2}{L^3}.$$

Putting everything together, we have

$$\begin{aligned}
&\mathbb{E}\left[\left(J^2 - \hat{J}^2\right)^2\right] \\
&\leq 4\gamma^{2T_J} R_{\max}^4 + 4\gamma^{T_J} R_{\max}^2 \frac{\zeta_2}{L} + \frac{\zeta_2^2}{L^2} \\
&\quad + 4\overline{G}_{0:T_J-1}^2 \frac{\zeta_2}{L} + \frac{2\zeta_2^2 + 4\zeta_3\overline{G}_{0:T_J-1}}{L^2} + \frac{\zeta_4 - 3\zeta_2^2}{L^3} \\
&\leq 4\gamma^{2T_J} R_{\max}^4 + 4(\overline{G}_{0:T_J-1}^2 + \gamma^{T_J} R_{\max}^2)\frac{\zeta_2}{L} + \frac{3\zeta_2^2 + 4|\zeta_3|R_{\max}}{L^2} + \frac{\zeta_4 - 3\zeta_2^2}{L^3} \\
&\leq 4\gamma^{2T_J} R_{\max}^4 + 4R_{\max}^2((1-\gamma^{T_J})^2 + \gamma^{T_J})\frac{\zeta_2}{L} + \frac{3\zeta_2^2 + 4|\zeta_3|R_{\max}}{L^2} + \frac{\zeta_4 - 3\zeta_2^2}{L^3} \\
&= 4\gamma^{2T_J} R_{\max}^4 + 4R_{\max}^2(1 + \gamma^{2T_J} - \gamma^{T_J})\frac{\zeta_2}{L} + \frac{3\zeta_2^2 + 4|\zeta_3|R_{\max}}{L^2} + \frac{\zeta_4 - 3\zeta_2^2}{L^3} \\
&\leq 4\gamma^{2T_J} R_{\max}^4 + 4R_{\max}^2\frac{\zeta_2}{L} + \frac{3\zeta_2^2 + 4|\zeta_3|R_{\max}}{L^2} + \frac{\zeta_4 - 3\zeta_2^2}{L^3}.
\end{aligned}$$

Furthermore, we can apply Lemma 4.2 and get

$$\begin{aligned}
\mathbb{E}\left[\left(J^2 - \hat{J}^2\right)^2\right] &\leq 4\gamma^{2T_J} R_{\max}^4 + \frac{4R_{\max}^4}{L} + \frac{3R_{\max}^4 + 4R_{\max}^4}{L^2} + \frac{2R_{\max}^4 + 3R_{\max}^4}{L^3} \\
&= 4\gamma^{2T_J} R_{\max}^4 + \left(\frac{4}{L} + \frac{7}{L^2} + \frac{5}{L^3}\right) R_{\max}^4.
\end{aligned}$$

□

Compared to the bounds in Section 4.3, the bounds in this section are concerned with the *squares* of the quantities $|J - \hat{J}|$ and $|J^2 - \hat{J}^2|$. This explains the bigger constants (e.g. $R_{\max}^4$ instead of $R_{\max}^2$) present in the derived bounds.

5.4 Critic's Analysis

In this section we derive the convergence rate of the critic (Algorithm 5.2) to the TD fixed point ω_J^*, where J is the true (normalized) expected return of the policy under evaluation. In our proof, we will, naturally, use the critic bound of [11] for the risk-neutral case[9]. We will not reiterate their proof here, but we can briefly mention the main idea of their approach. Suppose we are at iteration $t+1$ of the critic's algorithm, they start by bounding[10] $\mathbb{E}[\|\omega_{t+1} - \omega^*\|_2^2]$ in terms of the expected error at the previous iteration (i.e. $\mathbb{E}[\|\omega_t - \omega^*\|_2^2]$) and the expected[11] value of the squared norm of the

[9]Note that the risk-neutral case is a special case of our setting when λ is set to zero.

[10]Note that opposite to [11], we use ω for the parameters of the critic and θ for the parameters of the policy. This is done to keep the notation consistent with the previous chapters.

[11]Note that the performed updates are random due to the stochasticity of the sampling process.

difference between the performed update from ω_t to ω_{t+1} and its expected value at steady state. The latter quantity is then bounded in terms of the mixing properties of the MDP (see Assumption 5.4), the mini-batch size M, and again, $\mathbb{E}[\|\omega_t - \omega^*\|_2^2]$. By recursively repeating the same analysis on $\mathbb{E}[\|\omega_t - \omega^*\|_2^2]$ and the resulting terms, they obtain the following bound (under some conditions that we will mention later on the mini-batch size and the step-size):

$$\mathbb{E}[\|\omega_{T_c} - \omega^*\|_2^2] \leq \left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c} \|\omega_0 - \omega^*\|_2^2 + \left(\frac{2}{\lambda_A} + 2\beta\right) \frac{192(C_A^2 C_\omega^2 + C_b^2)[1 + (\kappa - 1)\rho]}{(1-\rho)\lambda_A M}.$$

The first term, which depends on the initial value of the parameter vector, decays geometrically with the number of iteration. The second term decays with a rate of $\frac{1}{M}$, where M is the size of the mini-batch of samples used at each iteration. Note that the step-size is kept constant across the iterations.

As we discussed before, for the reward transformation performed using any *fixed* estimate $\hat{J}$ of the expected return, we can directly use the bound above to establish the convergence rate to the TD fixed point under the this reward transformation (i.e. to $\omega_{\hat{J}}^*$). The next theorem uses this idea, along with (among other things) the bounds in Section 5.3, to establish the convergence rate of the critic to the true fixed point ω_J^* (i.e. under the true reward transformation) when $\hat{J}$ is learned using Algorithm 3.1.

Theorem 5.1 (Critic's Bound). *Suppose Assumptions 5.1 to 5.5 hold, and suppose we are given a policy π_θ (with normalized expected return J) and risk parameter λ. Suppose that a Monte-Carlo estimate $\hat{J}$ is obtained for π_θ using Algorithm* 3.1*, and then plugged into the Algorithm 5.2 which is run for T_c steps. Then, for $M \geq \left(\frac{2}{\lambda_A} + 2\beta\right) \frac{192 C_A^2[1+(\kappa-1)\rho]}{(1-\rho)\lambda_A}$ and $\beta \leq \min\left\{\frac{\lambda_A}{8C_A^2}, \frac{4}{\lambda_A}\right\}$, we have that*

$$\begin{aligned}\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_J^*\right\|_2^2\right] \leq{}& 4\|\omega_0 - \omega_J^*\|_2^2 \left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c} \\ &+ \left(\frac{2}{\lambda_A} + 2\beta\right) \frac{384(C_A^2 C_\omega^2 + C_b^2)[1 + (\kappa - 1)\rho]}{(1-\rho)\lambda_A M} \\ &+ \frac{2}{\bar{\sigma}^2}\left[1 + 2\left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c}\right] \xi_J,\end{aligned}$$

where $\omega_{T_c}^{\hat{J}}$ is the parameter vector obtained after T_c iterations of the algorithm while using $\hat{J}$ to perform the reward transformation, $\xi_J := 2\lambda^2 R_{\max}^4 \left(8\gamma^{2T_J} + \frac{8}{L} + \frac{7}{L^2} + \frac{5}{L^3}\right)$, $\bar{\sigma}$ is the smallest singular value of the matrix A, and the expectation is over both the Monte-Carlo estimation of $\hat{J}$ and the TD algorithm.

Proof. We begin by adding and subtracting $\omega_{\hat{J}}^*$, which is the TD fixed point when using $\hat{J}$. Note that, at this point, $\omega_{\hat{J}}^*$ is a random variable due to its dependence on $\hat{J}$.

$$\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_J^*\right\|_2^2\right] = \mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_{\hat{J}}^* + \omega_{\hat{J}}^* - \omega_J^*\right\|_2^2\right]$$

$$\leq 2\,\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_{\hat{J}}^*\right\|_2^2\right] + 2\,\mathbb{E}\left[\left\|\omega_{\hat{J}}^* - \omega_J^*\right\|_2^2\right]. \tag{5.6}$$

where the inequality follows from Lemma A.2.ii. Focusing on the first term, we have

$$\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_{\hat{J}}^*\right\|_2^2\right] = \mathbb{E}\left[\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_{\hat{J}}^*\right\|_2^2 \middle| \hat{J}\right]\right]. \tag{5.7}$$

For the inner expectation, as remarked before, we can apply the risk-neutral bound from theorem 4 in [11]. Namely for $M \geq \left(\frac{2}{\lambda_A} + 2\beta\right)\frac{192C_A^2[1+(\kappa-1)\rho]}{(1-\rho)\lambda_A}$ and $\beta \leq \min\left\{\frac{\lambda_A}{8C_A^2}, \frac{4}{\lambda_A}\right\}$, we have

$$\begin{aligned}
&\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_{\hat{J}}^*\right\|_2^2 \middle| \hat{J}\right] \\
&\leq \left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c}\left\|\omega_0 - \omega_{\hat{J}}^*\right\|_2^2 + \left(\frac{2}{\lambda_A} + 2\beta\right)\frac{192(C_A^2C_\omega^2 + C_b^2)[1+(\kappa-1)\rho]}{(1-\rho)\lambda_A M}.
\end{aligned}$$

Note that $\|\omega_0 - \omega_{\hat{J}}^*\|_2^2$ is only part that depends on $\hat{J}$ in the previous bound. Plugging back in (5.7), we get that

$$\begin{aligned}
&\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_{\hat{J}}^*\right\|_2^2\right] \\
&\leq \left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c}\mathbb{E}\left[\left\|\omega_0 - \omega_{\hat{J}}^*\right\|_2^2\right] + \left(\frac{2}{\lambda_A} + 2\beta\right)\frac{192(C_A^2C_\omega^2 + C_b^2)[1+(\kappa-1)\rho]}{(1-\rho)\lambda_A M} \\
&\leq \left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c}\mathbb{E}\left[\left\|\omega_0 - \omega_J^* + \omega_J^* - \omega_{\hat{J}}^*\right\|_2^2\right] + \left(\frac{2}{\lambda_A} + 2\beta\right)\frac{192(C_A^2C_\omega^2 + C_b^2)[1+(\kappa-1)\rho]}{(1-\rho)\lambda_A M} \\
&\leq 2\left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c}\|\omega_0 - \omega_J^*\|_2^2 + \left(\frac{2}{\lambda_A} + 2\beta\right)\frac{192(C_A^2C_\omega^2 + C_b^2)[1+(\kappa-1)\rho]}{(1-\rho)\lambda_A M} \\
&\quad + 2\left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c}\mathbb{E}\left[\left\|\omega_{\hat{J}}^* - \omega_J^*\right\|_2^2\right],
\end{aligned}$$

where the last inequality again follows from Lemma A.2.ii. Plugging back in (5.6), we get that

$$\begin{aligned}
&\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_J^*\right\|_2^2\right] \\
&\leq 4\left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c}\|\omega_0 - \omega_J^*\|_2^2 + \left(\frac{2}{\lambda_A} + 2\beta\right)\frac{384(C_A^2C_\omega^2 + C_b^2)[1+(\kappa-1)\rho]}{(1-\rho)\lambda_A M} \\
&\quad + \left[2 + 4\left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c}\right]\mathbb{E}\left[\left\|\omega_{\hat{J}}^* - \omega_J^*\right\|_2^2\right].
\end{aligned} \tag{5.8}$$

Thus, we only need to bound $\mathbb{E}\left[\left\|\omega_{\hat{J}}^* - \omega_J^*\right\|_2^2\right]$. We proceed as follows:

$$\mathbb{E}\left[\left\|\omega_{\hat{J}}^* - \omega_J^*\right\|_2^2\right] = \mathbb{E}\left[\left\|A^{-1}b(J) - A^{-1}b(\hat{J})\right\|_2^2\right]$$

$$\begin{aligned}
&= \mathbb{E}\left[\left\|A^{-1}\Big(b(J) - b(\hat{J})\Big)\right\|_2^2\right] \\
&\leq \frac{1}{\bar{\sigma}^2}\,\mathbb{E}\left[\left\|b(J) - b(\hat{J})\right\|_2^2\right], \qquad (5.9)
\end{aligned}$$

where $\bar{\sigma}$ is the smallest singular value of A, and the last inequality holds since, as demonstrated before, for an $m \times n$ matrix X and a vector $y \in \mathbb{R}^n$, $\|Xy\|_2^2 \leq \|X\|_2^2\|y\|_2^2$, where $\|X\|_2$ is the spectral norm of X. Furthermore, we used that, by Lemma A.1, $\left\|A^{-1}\right\|_2 = \frac{1}{\bar{\sigma}}$. Moving on, recall that μ_θ is the stationary distribution of the MDP when using policy π_θ. We then have that

$$\begin{aligned}
&\mathbb{E}\Big[\Big\|b(J) - b(\hat{J})\Big\|\Big]_2^2 \\
&= \mathbb{E}\left[\left\|\mathop{\mathbb{E}}_{\mu_\theta}\Big[\phi(s_t)R^\lambda(s_t,a_t,J)\Big] - \mathop{\mathbb{E}}_{\mu_\theta}\Big[\phi(s_t)R^\lambda(s_t,a_t,\hat{J})\Big]\right\|_2^2\right] \\
&= \mathbb{E}\left[\left\|\mathop{\mathbb{E}}_{\mu_\theta}\Big[\phi(s_t)\Big(R^\lambda(s_t,a_t,J) - R^\lambda(s_t,a_t,\hat{J})\Big)\Big]\right\|_2^2\right] \\
&= \mathbb{E}\left[\left\|\mathop{\mathbb{E}}_{\mu_\theta}\Big[\phi(s_t)\Big(2\lambda R(s_t,a_t)\Big(J - \hat{J}\Big) + \lambda\Big(\hat{J}^2 - J^2\Big)\Big)\Big]\right\|_2^2\right] \\
&= \mathbb{E}\left[\left\|2\lambda\Big(J - \hat{J}\Big)\mathop{\mathbb{E}}_{\mu_\theta}[\phi(s_t)R(s_t,a_t)] + \lambda\Big(\hat{J}^2 - J^2\Big)\mathop{\mathbb{E}}_{\mu_\theta}[\phi(s_t)]\right\|_2^2\right] \\
&\leq \mathbb{E}\left[2\left\|2\lambda\Big(J - \hat{J}\Big)\mathop{\mathbb{E}}_{\mu_\theta}[\phi(s_t)R(s_t,a_t)]\right\|_2^2 + 2\left\|\lambda\Big(\hat{J}^2 - J^2\Big)\mathop{\mathbb{E}}_{\mu_\theta}[\phi(s_t)]\right\|_2^2\right] \\
&= 8\lambda^2\,\mathbb{E}\left[\Big(J - \hat{J}\Big)^2\left\|\mathop{\mathbb{E}}_{\mu_\theta}[\phi(s_t)R(s_t,a_t)]\right\|_2^2\right] + 2\lambda^2\,\mathbb{E}\left[\Big(\hat{J}^2 - J^2\Big)^2\left\|\mathop{\mathbb{E}}_{\mu_\theta}[\phi(s_t)]\right\|_2^2\right] \\
&\leq 8\lambda^2 R_{\max}^2\,\mathbb{E}\left[\Big(J - \hat{J}\Big)^2\right] + 2\lambda^2\,\mathbb{E}\left[\Big(\hat{J}^2 - J^2\Big)^2\right], \qquad (5.10)
\end{aligned}$$

where the first inequality follows from Lemma A.2, and the last inequality follows (keeping in mind Assumptions 5.1.i, 5.2, and 5.3) since

$$\left\|\mathop{\mathbb{E}}_{\mu_\theta}[\phi(s_t)R(s_t,a_t)]\right\|_2^2 \leq \mathop{\mathbb{E}}_{\mu_\theta}\Big[\|\phi(s_t)R(s_t,a_t)\|_2^2\Big] \leq R_{\max}^2,$$

and

$$\left\|\mathop{\mathbb{E}}_{\mu_\theta}[\phi(s_t)]\right\|_2^2 \leq \mathop{\mathbb{E}}_{\mu_\theta}\Big[\|\phi(s_t)\|_2^2\Big] \leq 1.$$

Now, we can plug the results of Propositions 5.1 and 5.2 in inequality (5.10) to get

$$\begin{aligned}
\mathbb{E}\Big[\Big\|b(J) - b(\hat{J})\Big\|\Big]_2^2 &\leq 8\lambda^2 R_{\max}^2\,\mathbb{E}\left[\Big(J - \hat{J}\Big)^2\right] + 2\lambda^2\,\mathbb{E}\left[\Big(\hat{J}^2 - J^2\Big)^2\right] \\
&\leq 8\lambda^2 R_{\max}^2\left(\gamma^{2T_J} R_{\max}^2 + \frac{R_{\max}^2}{L}\right)
\end{aligned}$$

$$
\begin{aligned}
&+ 2\lambda^2\left(4\gamma^{2T_J}R_{\max}^4 + \left(\frac{4}{L}+\frac{7}{L^2}+\frac{5}{L^3}\right)R_{\max}^4\right)\\
&= 8\lambda^2\gamma^{2T_J}R_{\max}^4 + \frac{8\lambda^2}{L}R_{\max}^4\\
&\quad + 8\lambda^2\gamma^{2T_J}R_{\max}^4 + 2\lambda^2\left(\frac{4}{L}+\frac{7}{L^2}+\frac{5}{L^3}\right)R_{\max}^4\\
&= 16\lambda^2\gamma^{2T_J}R_{\max}^4 + 2\lambda^2\left(\frac{8}{L}+\frac{7}{L^2}+\frac{5}{L^3}\right)R_{\max}^4.
\end{aligned}
$$

Plugging back in (5.9), we get

$$
\begin{aligned}
\mathbb{E}\left[\left\|\omega_{\hat{J}}^* - \omega_J^*\right\|_2^2\right] &\le \frac{2\lambda^2}{\bar{\sigma}^2}\left(8\gamma^{2T_J}R_{\max}^4 + \left(\frac{8}{L}+\frac{7}{L^2}+\frac{5}{L^3}\right)R_{\max}^4\right)\\
&= \frac{\xi_J}{\bar{\sigma}^2},
\end{aligned}
$$

where $\xi_J := 2\lambda^2 R_{\max}^4\big(8\gamma^{2T_J} + \frac{8}{L} + \frac{7}{L^2} + \frac{5}{L^3}\big)$. We can now plug back the last result into (5.8) to get

$$
\begin{aligned}
\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_J^*\right\|_2^2\right] &\le 4\|\omega_0 - \omega_J^*\|_2^2\left(1-\frac{\lambda_A}{8}\beta\right)^{T_c}\\
&\quad + \left(\frac{2}{\lambda_A}+2\beta\right)\frac{384(C_A^2C_\omega^2 + C_b^2)[1+(\kappa-1)\rho]}{(1-\rho)\lambda_A M}\\
&\quad + \frac{2}{\bar{\sigma}^2}\left[1+2\left(1-\frac{\lambda_A}{8}\beta\right)^{T_c}\right]\xi_J.
\end{aligned}
$$

□

Note that the first two terms of the bound are (up to constants) the risk-neutral bound of [11]. The third term contains ξ_J, which quantifies the inaccuracy of $\hat{J}$, and decays by increasing L and T_J. By further inspecting the third term, we can see that ξ_J appears once on its own and once multiplied by a term that decays as the number of iterations of the critic increase. Another important thing to notice is that our assumption that $\hat{J}$ is learned using different samples than the ones collected by the critic is crucial for directly using the bound of [11] when fixing $\hat{J}$ (as in 5.7). This is because otherwise, the distribution of the performed updates of the critic differs when knowing $\hat{J}$, and the analysis of [11] is no longer applicable. The following corollary provides a detailed derivation of the sample complexity of the critic.

Corollary 5.1.1 (Critic's Complexity)**.** *Suppose we are again in the same setting of Theorem 5.1, and suppose the assumptions mentioned therein hold. Then, for a sufficiently small* $\epsilon > 0$*, if* $\beta \le \min\left\{\frac{\lambda_A}{8C_A^2}, \frac{4}{\lambda_A}\right\}$, $T_J \ge \frac{\log\left(\frac{192\lambda^2 R_{\max}^4}{\epsilon\bar{\sigma}^2}\right)}{2(1-\gamma)}$*, and* $L \ge \frac{576\lambda^2 R_{\max}^4}{\epsilon\bar{\sigma}^2}$*,* $T_c \ge \frac{8\log\left(\frac{24}{\epsilon}\|\omega_0-\omega_J^*\|_2^2\right)}{\lambda_A\beta}$*,* $M \ge \left(\frac{2}{\lambda_A}+2\beta\right)\frac{2304(C_A^2C_\omega^2+C_b^2)[1+(\kappa-1)\rho]}{(1-\rho)\lambda_A\epsilon}$*, then*

$$
\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_J^*\right\|_2^2\right] \le \epsilon,
$$

and the total sample complexity is

$$T_c M + L T_J = \mathcal{O}\big(\epsilon^{-1}\log\big(\epsilon^{-1}\big)\big).$$

Proof. By expanding and rearranging the bound in Theorem 5.1, we have that

$$\begin{aligned}\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_J^*\right\|_2^2\right] \le\; & 4\|\omega_0 - \omega_J^*\|_2^2\left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c} \\ & + \left(\frac{2}{\lambda_A} + 2\beta\right)\frac{384(C_A^2 C_\omega^2 + C_b^2)[1 + (\kappa - 1)\rho]}{(1-\rho)\lambda_A M} \\ & + \frac{32\lambda^2 R_{\max}^4}{\bar{\sigma}^2}\gamma^{2T_J} \\ & + \frac{4\lambda^2 R_{\max}^4}{\bar{\sigma}^2}\left(\frac{8}{L} + \frac{7}{L^2} + \frac{5}{L^3}\right) \\ & + \frac{64\lambda^2 R_{\max}^4}{\bar{\sigma}^2}\gamma^{2T_J}\left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c} \\ & + \frac{8\lambda^2 R_{\max}^4}{\bar{\sigma}^2}\left(\frac{8}{L} + \frac{7}{L^2} + \frac{5}{L^3}\right)\left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c}.\end{aligned}$$

Note that $\left(1 - \frac{\lambda_A}{8}\beta\right)^{T_c} \le e^{-\frac{\lambda_A}{8}\beta T_c}$. This holds since $(1-x) \le e^{-x}$, and if $x \le 1$, then $(1-x)^r \le e^{-rx}$ for $r \ge 0$. The claim then follows since $\beta < \frac{8}{\lambda_A}$ and $T_c \ge 0$. By a similar argument, $\gamma^{2T_J} = (1-(1-\gamma))^{2T_J} \le e^{-2(1-\gamma)T_J}$. Plugging back these bounds, we get

$$\begin{aligned}\mathbb{E}\left[\left\|\omega_{T_c}^{\hat{J}} - \omega_J^*\right\|_2^2\right] \le\; & 4\|\omega_0 - \omega_J^*\|_2^2\, e^{-\frac{\lambda_A}{8}\beta T_c} \\ & + \left(\frac{2}{\lambda_A} + 2\beta\right)\frac{384(C_A^2 C_\omega^2 + C_b^2)[1 + (\kappa - 1)\rho]}{(1-\rho)\lambda_A M} \\ & + \frac{32\lambda^2 R_{\max}^4}{\bar{\sigma}^2}\, e^{-2(1-\gamma)T_J} \\ & + \frac{4\lambda^2 R_{\max}^4}{\bar{\sigma}^2}\left(\frac{8}{L} + \frac{7}{L^2} + \frac{5}{L^3}\right) \\ & + \frac{64\lambda^2 R_{\max}^4}{\bar{\sigma}^2}\, e^{-2(1-\gamma)T_J}\, e^{-\frac{\lambda_A}{8}\beta T_c} \\ & + \frac{8\lambda^2 R_{\max}^4}{\bar{\sigma}^2}\left(\frac{8}{L} + \frac{7}{L^2} + \frac{5}{L^3}\right) e^{-\frac{\lambda_A}{8}\beta T_c}.\end{aligned}$$

To bound the whole expression by ϵ, we can bound each of the six terms by $\frac{\epsilon}{6}$. This can be achieved for each term if

Term 1

$$T_c \ge \frac{8\log\left(\frac{24}{\epsilon}\|\omega_0 - \omega_J^*\|_2^2\right)}{\lambda_A\beta}$$

Term 2

$$M \geq \left(\frac{2}{\lambda_A} + 2\beta\right)\frac{2304(C_A^2 C_\omega^2 + C_b^2)[1 + (\kappa - 1)\rho]}{(1-\rho)\lambda_A \epsilon}$$

Term 3

$$T_J \geq \frac{\log\left(\frac{192\lambda^2 R_{\max}^4}{\epsilon\bar{\sigma}^2}\right)}{2(1-\gamma)}$$

Term 4

$$L \geq \max\left\{\frac{576\lambda^2 R_{\max}^4}{\epsilon\bar{\sigma}^2}, \sqrt{\frac{504\lambda^2 R_{\max}^4}{\epsilon\bar{\sigma}^2}}, \sqrt[3]{\frac{360\lambda^2 R_{\max}^4}{\epsilon\bar{\sigma}^2}}\right\}$$

Term 5

$$T_c \geq \frac{8\log\left(\frac{\sqrt{6}}{\sqrt{\epsilon}}\right)}{\lambda_A \beta},\ T_J \geq \frac{\log\left(\frac{64\sqrt{6}\lambda^2 R_{\max}^4}{\sqrt{\epsilon}\bar{\sigma}^2}\right)}{2(1-\gamma)}$$

Term 6

$$T_c \geq \frac{8log\left(\frac{\sqrt{6}}{\sqrt{\epsilon}}\right)}{\lambda_A \beta}, L \geq \max\left\{\frac{192\sqrt{6}\lambda^2 R_{\max}^4}{\sqrt{\epsilon}\bar{\sigma}^2}, \sqrt{\frac{168\sqrt{6}\lambda^2 R_{\max}^4}{\sqrt{\epsilon}\bar{\sigma}^2}}, \sqrt[3]{\frac{120\sqrt{6}\lambda^2 R_{\max}^4}{\sqrt{\epsilon}\bar{\sigma}^2}}\right\}$$

Note that there are multiple conditions on some parameters. However, if ϵ is sufficiently small, it is enough that $T_c \geq \frac{8\log\left(\frac{24}{\epsilon}\|\omega_0 - \omega_J^*\|_2^2\right)}{\lambda_A \beta}$, $M \geq \left(\frac{2}{\lambda_A} + 2\beta\right)\frac{2304(C_A^2 C_\omega^2 + C_b^2)[1+(\kappa-1)\rho]}{(1-\rho)\lambda_A\epsilon}$, $T_J \geq \frac{\log\left(\frac{192\lambda^2 R_{\max}^4}{\epsilon\bar{\sigma}^2}\right)}{2(1-\gamma)}$, and $L \geq \frac{576\lambda^2 R_{\max}^4}{\epsilon\bar{\sigma}^2}$. Thus, the sample complexity is given by

$$T_c M + L T_J = \mathcal{O}\left(\frac{1}{\epsilon}\log\left(\frac{1}{\epsilon}\right)\right) + \mathcal{O}\left(\frac{1}{\epsilon}\log\left(\frac{1}{\epsilon}\right)\right) = \mathcal{O}\left(\frac{1}{\epsilon}\log\left(\frac{1}{\epsilon}\right)\right).$$

□

The previous corollary essentially implies that the sample complexity of our risk-averse critic is not worsened compared to its risk-neutral counterpart, even though we are using an extra batch of samples to estimate the expected return. It is an interesting direction to investigate how the complexity will be affected if the mini-batch TD procedure used the same batch of data that was used to estimate the expected return.

5.5 Actor's Analysis

In this section, we move over to analyzing the actor procedure in Algorithm 5.1. Since the actor relies on the critic for the estimation of the gradient, the convergence of the actor to a stationary point of η_θ naturally relies on the accuracy of the critic. Undoubtedly then, the results of the previous section will be utilized in the analysis of the actor. However, the analysis of the last section was only concerned with how far the critic was from the TD fixed point. We will thus need an additional notion to describe the approximation error incurred due to not only using a linear function, but also for using the semi-gradient TD algorithm, whose fixed point is, in general, different from the best approximation in our space of candidate function (as discussed in Subsection 2.1.2). Thus, we define the following quantity

$$\xi_{appr} := \max_{\theta \in \mathbb{R}^{d_\theta}} \mathop{\mathbb{E}}_{s \sim d_{\mu_0, \pi_\theta}(\cdot)} \left[\left| V^{\lambda}_{\pi_\theta}(s) - \phi(s)^\intercal \omega^*_{J_\theta} \right|^2 \right],$$

which represents, for the worst possible policy, the mean squared error, over the discounted state distribution, between the true transformed value function and the approximated transformed value function at the TD fixed point.

Remember that our aim is to bound $\mathbb{E}\left[\left\|\nabla\eta(\theta_{\hat{T}})\right\|_2^2\right]$. To do this, we will extend the analysis in [11] to our risk-averse case. We first define the following quantities, which will help us in the analysis:

- the TD-error[12] $\delta_\omega(s,a,s') = R^\lambda(s,a,J) + \gamma\phi(s')^\top\omega - \phi(s)^\top\omega$ which employs the exact expected return J;
- the *approximated* TD-error $\hat{\delta}_\omega(s,a,s') = R^\lambda(s,a,\hat{J}) + \gamma\phi(s')^\top\omega - \phi(s)^\top\omega$ which employs, instead, the the Monte-Carlo current estimate of the expected return $\hat{J}$;
- $v_t(\omega,\theta) = \frac{1}{B}\sum_{i=0}^{B-1} \delta_\omega(s_{t,i}, a_{t,i}, s'_{t,i+1})\psi_{\theta_t}(s_{t,i}, a_{t,i})$, which would have been the estimated gradient at time t (using a critic with parameters ω) if we had access to the true J_θ;
- $\hat{v}_t(\omega,\theta) = \frac{1}{B}\sum_{i=0}^{B-1} \hat{\delta}_\omega(s_{t,i}, a_{t,i}, s'_{t,i+1})\psi_{\theta_t}(s_{t,i}, a_{t,i})$, which is the estimated gradient at time t (using a critic with parameters ω) based on $\hat{J}$;
- $A_\omega(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)}[\delta_\omega(s,a,s')]$, which is the expected value of the TD-error δ_ω at a given state-action pair when the next state is sampled from the transition kernel of the original MDP;
- $g(\omega,\theta) = \mathbb{E}_{\substack{s \sim d_{\mu_0,\pi_\theta}(\cdot) \\ a \sim \pi_\theta(\cdot|s)}}[A_\omega(s,a)\psi_\theta(s,a)]$, which is the expectation of the estimated gradient when using a critic with parameter vector ω.

Next, we prove two propositions, which will be combined to bound the expectation on the gradient norm.

Proposition 5.3. *Suppose Assumptions 5.1 to 5.5 hold, then:*

$$\left(\frac{\alpha}{2} - 2L_\eta\alpha^2\right)\|\nabla\eta(\theta_t)\|_2^2 \le \eta(\theta_{t+1}) - \eta(\theta_t) + \left(\frac{\alpha}{2} + 2L_\eta\alpha^2\right)\|\hat{v}_t(\omega_t,\theta_t) - \nabla\eta(\theta_t)\|_2^2.$$

[12]Note that the $\delta_\omega(s,a,s')$ and $\hat{\delta}_\omega(s,a,s')$ do depend on the current policy since they depend on its expected return, or an estimate of it. However, we do not explicitly express this dependence as to not burden the notation since it is usually clear from the context.

Proof. By applying the Mean-Value Theorem, for some $0 \leq \Delta \leq 1$ there is some $\tilde{\theta} = \Delta\theta_t + (1-\Delta)\theta_{t+1}$ such that:

$$\begin{aligned}\eta(\theta_{t+1}) &= \eta(\theta_t) + (\theta_{t+1}-\theta_t)^\top \nabla\eta(\tilde{\theta}) = \eta(\theta_t) + (\theta_{t+1}-\theta_t)^\top \nabla\eta(\tilde{\theta}) \pm (\theta_{t+1}-\theta_t)^\top \nabla\eta(\theta_t) \\ &= \eta(\theta_t) + (\theta_{t+1}-\theta_t)^\top \left(\nabla\eta(\tilde{\theta}) - \nabla\eta(\theta_t)\right) + (\theta_{t+1}-\theta_t)^\top \nabla\eta(\theta_t).\end{aligned}$$

By using Cauchy-Schwarz we also have:

$$\begin{aligned}(\theta_{t+1}-\theta_t)^\top \left(\nabla\eta(\tilde{\theta}) - \nabla\eta(\theta_t)\right) &\geq -\|\theta_{t+1}-\theta_t\|_2 \|\nabla\eta(\tilde{\theta}) - \nabla\eta(\theta_t)\|_2 \\ &\geq -L_\eta \|\theta_{t+1}-\theta_t\|_2 \|\tilde{\theta}-\theta_t\|_2 \\ &\geq -L_\eta \|\theta_{t+1}-\theta_t\|_2^2\end{aligned}$$

where we also used that the gradient of η is Lipschitz (Lemma B.4).

We exploit this relationship in the previous equation, together with the definition of the policy parameters update:

$$\begin{aligned}\eta(\theta_{t+1}) &\geq \eta(\theta_t) - L_\eta \|\theta_{t+1}-\theta_t\|_2^2 + (\theta_{t+1}-\theta_t)^\top \nabla\eta(\theta_t) \\ &= \eta(\theta_t) - \alpha^2 L_\eta \|\hat{v}_t(\omega_t,\theta_t)\|_2^2 + \alpha \hat{v}_t(\omega_t,\theta_t)^\top \nabla\eta(\theta_t) \\ &= \eta(\theta_t) - \alpha^2 L_\eta \|\hat{v}_t(\omega_t,\theta_t) \pm \nabla\eta(\theta_t)\|_2^2 + \alpha\langle \hat{v}_t(\omega_t,\theta_t) \pm \nabla\eta(\theta_t), \nabla\eta(\theta_t)\rangle \\ &\overset{(1)}{\geq} \eta(\theta_t) - 2\alpha^2 L_\eta \|\nabla\eta(\theta_t)\|_2^2 - 2\alpha^2 L_\eta \|\hat{v}_t(\omega_t,\theta_t) - \nabla\eta(\theta_t)\|_2^2 + \\ &\quad + \alpha\|\nabla\eta(\theta_t)\|_2^2 + \alpha\langle \hat{v}_t(\omega_t,\theta_t) - \nabla\eta(\theta_t), \nabla\eta(\theta_t)\rangle \\ &\overset{(2)}{\geq} \eta(\theta_t) - 2\alpha^2 L_\eta \|\nabla\eta(\theta_t)\|_2^2 - 2\alpha^2 L_\eta \|\hat{v}_t(\omega_t,\theta_t) - \nabla\eta(\theta_t)\|_2^2 + \\ &\quad + \alpha\|\nabla\eta(\theta_t)\|_2^2 - \frac{\alpha}{2}\|\hat{v}_t(\omega_t,\theta_t) - \nabla\eta(\theta_t)\|_2^2 - \frac{\alpha}{2}\|\nabla\eta(\theta_t)\|_2^2,\end{aligned}$$

where in the last two steps we used, respectively, Lemma A.2.ii and Lemma A.2.i in (1) and (2). By re-ordering terms we obtain the desired result. □

The last term in the bound of the last proposition represents how far the estimated gradient is from the true one. Mirroring [11], the next proposition bounds the expected value of this quantity.

Proposition 5.4. *Suppose Assumptions 5.1 to 5.5 hold, and let $\mathcal{F}_t$ be the filtration on the samples up to iteration t:*

$$\begin{aligned}\mathbb{E}\left[\|\hat{v}_t(\omega_t,\theta_t) - \nabla\eta(\theta_t)\|_2^2 | \mathcal{F}_t\right] \leq &\frac{24(R_{\lambda,\max} + 2C_\omega)^2[1+(k-1)\rho]}{B(1-\rho)} \\ &+ 48\lambda^2 R_{\max}^2 \mathbb{E}\left[|J-\hat{J}|^2|\mathcal{F}_t\right] + 12\lambda^2 \mathbb{E}\left[|\hat{J}^2 - J^2|^2|\mathcal{F}_t\right] \\ &+ 24\|\omega^*_{J_t} - \omega_t\|_2^2 + 12\ \xi_{appr},\end{aligned}$$

where J_t is short for $J_{\pi_{\theta_t}}$, and $\omega^*_{J_t}$ is the TD fixed point for the transformed value function of policy π_{θ_t}.

Proof. Consider $\|\hat{v}_t(\omega_t, \theta_t) - \nabla\eta(\theta_t)\|_2^2$, we can decompose it in the following way (followed by an application of Lemma A.2.ii):

$$\begin{aligned}
&\|\hat{v}_t(\omega_t, \theta_t) - \nabla\eta(\theta_t)\|_2^2 \\
&= \left\|\hat{v}_t(\omega_t, \theta_t) \pm v_t(\omega^*_{J_t}, \theta_t) \pm g(\omega^*_{J_t}, \theta_t) - \nabla\eta(\theta_t)\right\|_2^2 \\
&\leq 3\underbrace{\left\|\hat{v}_t(\omega_t, \theta_t) - v_t(\omega^*_{J_t}, \theta_t)\right\|_2^2}_{(a)} + 3\left\|v_t(\omega^*_{J_t}, \theta_t) - g(\omega^*_{J_t}, \theta_t)\right\|_2^2 + 3\underbrace{\left\|g(\omega^*_{J_t}, \theta_t) - \nabla\eta(\theta_t)\right\|_2^2}_{(b)}.
\end{aligned} \tag{5.11}$$

We now focus on (a):

$$\begin{aligned}
&\left\|\hat{v}_t(\omega_t, \theta_t) - v_t(\omega^*_{J_t}, \theta_t)\right\|_2^2 \\
=&\left\|\frac{1}{B}\sum_{i=0}^{B-1} \psi_{\theta_t}(s_{t,i}, a_{t,i})\left[\hat{\delta}_{\omega_t}(s_{t,i}, a_{t,i}, s'_{t,i+1}) - \delta_{\omega^*_{J_t}}(s_{t,i}, a_{t,i}, s'_{t,i+1})\right]\right\|_2^2 \\
\leq&\frac{1}{B}\sum_{i=0}^{B-1} \underbrace{\|\psi_{\theta_t}(s_{t,i}, a_{t,i})\|_2^2}_{\leq C_\psi = 1}\left|\hat{\delta}_{\omega_t}(s_{t,i}, a_{t,i}, s'_{t,i+1}) - \delta_{\omega^*_{J_t}}(s_{t,i}, a_{t,i}, s'_{t,i+1})\right|^2 \\
\leq&\frac{1}{B}\sum_{i=0}^{B-1}\left|R^\lambda(s_{t,i}, a_{t,i}, \hat{J}) - R^\lambda(s_{t,i}, a_{t,i}, J) + \gamma\left(\phi(s'_{t,i+1})^\top\omega_t - \phi(s'_{t,i+1})^\top\omega^*_{J_t}\right) + \right. \\
&\qquad\left. + \left(\phi(s_{t,i})^\top\omega^*_{J_t} - \phi(s_{t,i})^\top\omega_t\right)\right|^2 \\
\overset{(1)}{\leq}&\frac{1}{B}\sum_{i=0}^{B-1} 2\left|R^\lambda(s_{t,i}, a_{t,i}, \hat{J}) - R^\lambda(s_{t,i}, a_{t,i}, J)\right|^2 + 2\left|(\gamma\phi(s'_{t,i+1}) - \phi(s_{t,i}))^\top(\omega_t - \omega^*_{J_t})\right|^2 \\
\overset{(2)}{\leq}&\frac{1}{B}\sum_{i=0}^{B-1} 2\left|R^\lambda(s_{t,i}, a_{t,i}, \hat{J}) - R^\lambda(s_{t,i}, a_{t,i}, J)\right|^2 + 8\left\|\omega^*_{J_t} - \omega_t\right\|_2^2 \\
\overset{(3)}{=}&\frac{1}{B}\sum_{i=0}^{B-1} 2\lambda^2\left|2R(s_{t,i}, a_{t,i})(J - \hat{J}) + \hat{J}^2 - J^2\right|^2 + 8\left\|\omega^*_{J_t} - \omega_t\right\|_2^2 \\
\overset{(4)}{\leq}&16\lambda^2 R^2_{\max}|J - \hat{J}|^2 + 4\lambda^2|\hat{J}^2 - J^2|^2 + 8\|\omega^*_{J_t} - \omega_t\|_2^2.
\end{aligned}$$

where (1) is an application of Lemma A.2.ii, (2) is due to Cauchy–Schwarz, Lemma A.2.ii, and Assumption 5.3.ii, (3) to definition of R^λ, and in (4) Lemma A.2.ii is applied again.

We can then exploit results from Theorem 5 in [11], to bound (b) as:

$$\left\|g(\omega^*_{J_t}, \theta_t) - \nabla\eta(\theta_t)\right\|_2^2 \leq 4\xi_{appr}.$$

Substituting back to inequality (5.11) and taking the expectation w.r.t. the filtration $\mathcal{F}_t$, we get:

$$\mathbb{E}\left[\|\hat{v}_t(\omega_t, \theta_t) - \nabla\eta(\theta_t)\|_2^2|\mathcal{F}_t\right] \leq 3\,\mathbb{E}\left[\|v_t(\omega^*, \theta_t) - g(\omega^*, \theta_t)\|_2^2|\mathcal{F}_t\right]$$

$$+ 48\lambda^2 R_{max}^2 \mathbb{E}\left[|J-\hat{J}|^2|\mathcal{F}_t\right] + 12\lambda^2 \mathbb{E}\left[|\hat{J}^2 - J^2|^2|\mathcal{F}_t\right]$$
$$+ 24\|\omega^*_{J_t} - \omega_t\|_2^2 + 12\ \xi_{appr}.$$

To bound the conditional expectation on the RHS, we follow again the proof in [11] to have:

$$\mathbb{E}\left[\|v_t(\omega^*, \theta_t) - g(\omega^*, \theta_t)\|_2^2|\mathcal{F}_t\right] \le \frac{8(R_{\lambda,\max} + 2C_\omega)^2(1+(k-1)\rho)}{B(1-\rho)}. \tag{5.12}$$

□

Theorem 5.2 (Actor's Bound)**.** *Suppose Assumptions 5.1 to 5.5 hold and let* $\alpha = \frac{1}{8L_\eta}$*, then we have:*

$$\mathbb{E}\left[\left\|\nabla\eta(\theta_{\hat{T}})\right\|_2^2\right] \le \frac{64L_\eta R_{\lambda,\max}}{T} + \xi_{distr} + 18\xi_J + 72\frac{\sum_{t=0}^{T-1}\mathbb{E}[\|\omega^*_{J_t} - \omega_t\|_2^2]}{T} + 36\xi_{appr},$$

where

$$\xi_{distr} := \frac{72(R_{\lambda,\max} + 2C_\omega)^2(1+(k-1)\rho)}{B(1-\rho)}.$$

Proof. Taking the conditioned expectation on the result of Proposition 5.3 and plugging what we obtained with Proposition 5.4 we obtain the following:

$$\begin{aligned}
&\left(\frac{\alpha}{2} - 2L_\eta\alpha^2\right)\mathbb{E}\left[\|\nabla\eta(\theta_t)\|_2^2|\mathcal{F}_t\right] \\
&\le \mathbb{E}\left[\eta(\theta_{t+1})|\,\mathcal{F}_t]\right] - \eta(\theta_t) + \left(\frac{\alpha}{2} + 2L_\eta\alpha^2\right)\left[\frac{24(R_{\lambda,\max} + 2C_\omega)^2[1+(k-1)\rho]}{B(1-\rho)} + \right. \\
&\quad \left. + 48\lambda^2 R_{\max}^2 \mathbb{E}\left[|J-\hat{J}|^2|\mathcal{F}_t\right] + 12\lambda^2\mathbb{E}\left[|\hat{J}^2 - J^2|^2|\mathcal{F}_t\right] + 24\|\omega^*_{J_t} - \omega_t\|_2^2 + 12\ \xi_{appr}\ \right] \\
&\le \mathbb{E}\left[\eta(\theta_{t+1})|\,\mathcal{F}_t]\right] - \eta(\theta_t) + \left(\frac{\alpha}{2} + 2L_\eta\alpha^2\right)\left[\frac{24(R_{\lambda,\max} + 2C_\omega)^2(1+(k-1)\rho)}{B(1-\rho)} + \right. \\
&\quad + 48\lambda^2 R_{\max}^2\left(\gamma^{2T_J}R_{\max}^2 + \frac{R_{\max}^2}{L}\right) + 12\lambda^2\left(4\gamma^{2T_J}R_{\max}^4 + \left(\frac{4}{L} + \frac{7}{L^2} + \frac{5}{L^3}\right)R_{\max}^4\right) + \\
&\quad \left. + 24\|\omega^*_{J_t} - \omega_t\|_2^2 + 12\ \xi_{appr}\ \right],
\end{aligned}$$

We let $\alpha = \frac{1}{8L_\eta}$ and we multiply both sides by $32L_\eta$ to get:

$$\begin{aligned}
\mathbb{E}\left[\|\nabla\eta(\theta_t)\|_2^2|\mathcal{F}_t\right] \le\ & 32L_\eta\left(\mathbb{E}\left[\eta(\theta_{t+1})|\mathcal{F}_t]\right] - \eta(\theta_t)\right) + \xi_{distr} + 18\xi_J \\
& + 72\|\omega^*_{J_t} - \omega_t\|_2^2 + 36\xi_{appr},
\end{aligned}$$

with

$$\xi_{distr} := \frac{72(R_{\lambda,\max} + 2C_\omega)^2(1+(k-1)\rho)}{B(1-\rho)},$$

which bounds the variance of the mini-batch estimate of the gradient if the critic was at the TD fixed point, while ξ_J, the error arising from the expected return estimation, has been already defined in Theorem 5.1.

We take the expectation w.r.t. $\mathcal{F}_t$ to both sides to yield:

$$\mathbb{E}\left[\|\nabla\eta(\theta_t)\|_2^2\right] \leq 32L_\eta(\mathbb{E}\left[\eta(\theta_{t+1})\right] - \mathbb{E}\left[\eta(\theta_t)\right]) + \xi_{distr} + 18\xi_J \\ + 72\,\mathbb{E}[\|\omega^*_{J_t} - \omega_t\|_2^2] + 36\xi_{appr}.$$

Taking the summation of the last result over $t = 0, \ldots, T-1$ and dividing both sides by T gives:

$$\begin{aligned}\mathbb{E}\left[\|\nabla\eta(\theta_{\hat{T}})\|_2^2\right] &= \frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\|\nabla\eta(\theta_t)\|_2^2\right] \\ &\leq 32L_\eta\frac{\mathbb{E}\left[\eta(\theta_T)\right] - \eta(\theta_0)}{T} + \xi_{distr} + 18\xi_J \\ &\quad + 72\frac{\sum_{t=0}^{T-1}\mathbb{E}[\|\omega^*_{J_t} - \omega_t\|_2^2]}{T} + 36\xi_{appr} \\ &\leq \frac{64L_\eta R_{\lambda,\max}}{T} + \xi_{distr} + 18\xi_J + 72\frac{\sum_{t=0}^{T-1}\mathbb{E}[\|\omega^*_{J_t} - \omega_t\|_2^2]}{T} + 36\xi_{appr}.\end{aligned}$$

□

The bound in Theorem 5.2 is made up of five terms. The first is an error term that decays as the number of actor iterations increases. The second term represents, as mentioned before, the error due to the variance of the mini-batch estimate of the gradient if the critic was at the TD fixed point, and it decays by increasing the mini-batch size. The third term represents the error due to the inaccuracy of the estimated expected return ($\hat{J}$), and it decays by increasing L and T_J. The fourth term is the average (across iterations) of how far we expect the critic to be from the TD fixed point. Lastly, the fifth term is the only irreducible one as it expresses the approximation error due to using temporal difference with linear function approximation in the critic. Compared to the bound in [11], our bound assumes a similar form (albeit some of the quantities are naturally defined differently in our setting), except for the addition of the ξ_J term. The following corollary derives the sample complexity of Algorithm 5.1.

Corollary 5.2.1 (Actor's Complexity)**.** *Suppose we are in the same setting of Theorem 5.2, and assume that the parameters used in the critic are conditioned as to make* $\mathbb{E}\left[\|\omega_t - \omega^*_{J_t}\|_2^2\right] \leq \frac{\epsilon}{360}$ *for all* $t = 0, \ldots, T-1$*. Then, if additionally*

- $T \geq \frac{320L_\eta(R_{\max}+4\lambda R^2_{\max})}{\epsilon}$,
- $B \geq \frac{360((R_{\max}+4\lambda R^2_{\max})+2C_\omega)^2(1+(k-1)\rho)}{(1-\rho)\epsilon}$,
- $T_J \geq \frac{\log\left(\frac{1440\lambda^2 R^4_{max}}{\epsilon}\right)}{2(1-\gamma)}$,

- $L \geq \frac{3600\lambda^2 R_{\max}^4}{\epsilon}$,

we have that

$$\mathbb{E}\left[\left\|\nabla\eta(\omega_{\hat{T}})\right\|_2^2\right] \leq \epsilon + \mathcal{O}(\xi_{appr}),$$

with the total sample complexity given by:

$$T((2-\gamma)B + MT_c + LT_J) = \mathcal{O}\left(\epsilon^{-2}\log\left(\epsilon^{-1}\right)\right).$$

Proof. In order to compute the different contributions to sample complexity, we will split the error bound obtained in Theorem 5.2 in its components. We then bound the components in the following way:

- $\frac{64L_\eta(R_{\max}+4\lambda R_{\max}^2)}{T} \leq \epsilon_1$,
- $\frac{72(R_{\max}+4\lambda R_{\max}^2+2C_\omega)^2(1+(k-1)\rho)}{B(1-\rho)} \leq \epsilon_2$,
- $288\lambda^2 R_{max}^4\gamma^{2T_J} \leq 288\lambda^2 R_{max}^4 e^{-2(1-\gamma)T_J} \leq \epsilon_3$,
- $36\lambda^2 R_{max}^4(\frac{8}{L}+\frac{7}{L^2}+\frac{5}{L^3}) \overset{L>1}{\leq} 36\lambda^2 R_{max}^4(\frac{20}{L}) \leq \epsilon_4$,
- $72\frac{\sum_{t=0}^{T-1}\mathbb{E}[\|\omega_{J_t}^*-\omega_t\|_2^2]}{T} \leq \epsilon_5$,

where we have split $18\xi_J$ in two parts, and we have ignored the approximation error ξ_{appr}, which cannot be reduced with more samples. We set, then, each ϵ_i to $\frac{\epsilon}{5}$. Rearranging terms in each inequality, we obtain then, by the conditions on the parameters indicated in the statement[13], the desired error. In order to obtain it, the following sample complexity is needed:

$$T((2-\gamma)B+MT_c+LT_J) = \mathcal{O}\left(\frac{1}{\epsilon}\left(\frac{1}{\epsilon}+\frac{1}{\epsilon}\log\left(\frac{1}{\epsilon}\right)+\frac{1}{\epsilon}\log\left(\frac{1}{\epsilon}\right)\right)\right) = \mathcal{O}\left(\epsilon^{-2}\log\left(\epsilon^{-1}\right)\right)$$

where the $(2-\gamma)$ extra factor is due to the actor sampling process, which needs to sample twice at each restart, which can happen at each step with probability $1-\gamma$. $\square$

This last result shows that, for the whole actor-critic algorithm, the sample complexity is still not worsened compared to the risk-neutral version. As remarked before, an interesting direction would be to explore the effects on sample complexity resulting from adapting different sampling schemes designed to, one way or another, reuse samples across the different modules of the algorithm.

[13] And the conditions adapted from Corollary 5.1.1 needed to make $\mathbb{E}[\|\omega_t - \omega_{J_t}^*\|_2^2] \leq \frac{\epsilon}{360}$ (instead of just ϵ) for all $t = 0, \dots, T-1$.

Chapter 6

Experiments

In this chapter, we apply some of the proposed algorithms to a simple environment with the purpose of illustrating the soundness and the performance of the algorithms. The environment we consider is a chain walk with a reward structure that leads to a trade-off between optimality in expectation and the variance of the rewards. The purpose of using this environment is to test both our policy evaluation methods and our actor-critic algorithm.

The considered chain walk consists of 15 states. At any state, the agent can choose to move to either the left or the right. With probability 0.85, the agent moves according to its will (except at the edges, where the state does not change if the agent attempts to move outside the chain), and with probability 0.15, the agents moves in the opposite direction. The received reward at each step is a deterministic function of the state. Figure 6.1 shows a schematic representation of the environment, where the reward associated to each state is indicated. This MDP is intended to be executed in a continuing

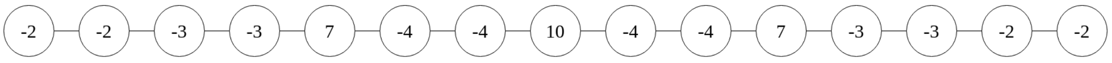

Figure 6.1: The risky chain walk environment.

manner, not in an episodic one. The central region is intended to be the preferable one for a risk-neutral agent since the central state gives off a large reward of 10, even though it surrounded by states that give a reward of -4. The regions a little bit off the center are in some sense similar to the central region, however they exhibit less variability between the high reward of 7 and the surrounding rewards of -3 and -4. The extreme ends of the chain exhibit the least variability of the rewards, although the rewards in these region are only negative. The initial state distribution μ_0 is uniform over the states. In the following, we will illustrate the performance of our actor critic algorithm at different values of lambda. Moreover, for a chosen policy (picked during the training process) at each value of lambda, we illustrate the performance of our policy evaluation methods.

6.1 Policy Optimization

We will test our actor-critic algorithm (Algorithm 5.1) in the environment at three different value of lambda to illustrate the performance at different levels of risk aversion. Namely we consider lambda = 0, 0.05, 0.07. We will use softmax policies, where the weight of each action is a linear function of the state. The features we use for the states are Gaussian radial basis functions with 3 means spread uniformly over the state space, and a width of 6. The critic is naturally learned according to Algorithm 5.2, and it uses the same features as the ones used by the policy. We also note that the learning processes of the expected return, the critic, and the actor all use their own batch of samples. The following is a summary of the parameters of the problem and the algorithm:

- Discount factor: $\gamma = 0.9$.
- Number of trajectories for estimating J: $L = 70$.
- Lengths of the trajectories for estimating J: $T_J = 70$.
- Critic batch size: $M = 10$.
- Number of critic iterations: $T_c = 50$.
- Critic step-size: $\beta = 0.1$
- Actor batch size: $B = 30$.
- Number of actor iterations: $T = 150$.
- Actor step-size: $\alpha = 0.05$

Risk-Neutral Objective ($\lambda = 0$)

At $\lambda = 0$, we seek a policy that maximizes J, regardless of the effect on the reward volatility. Figure 6.2 shows the performance of the actor-critic algorithm in the risk-neutral case, average over 10 runs. The first figure on the left shows the progress of the (un-normalized) J when the samples increase (i.e. with more iterations). The second and the third figures represent, respectively, the progress of the reward volatility and the mean volatility across the iterations. As expected, the performance (i.e. J) increases as we use more samples, but this comes at the cost of also increasing the reward volatility. Note that since $\lambda = 0$, the mean volatility is the same as J, however the difference in scale is because the mean volatility is normalized, while the values we plot for J are not. We will see an example of how risk-neutral policies act in the next section.

Risk-Averse Objective ($\lambda = 0.05$)

We know move to a risk-averse objective by setting $\lambda = 0.05$. Figure 6.3 shows the performance of the actor-critic algorithm for this case, also averaged over 10 runs. We can see from the figure that the algorithm still achieves higher values of J as the samples

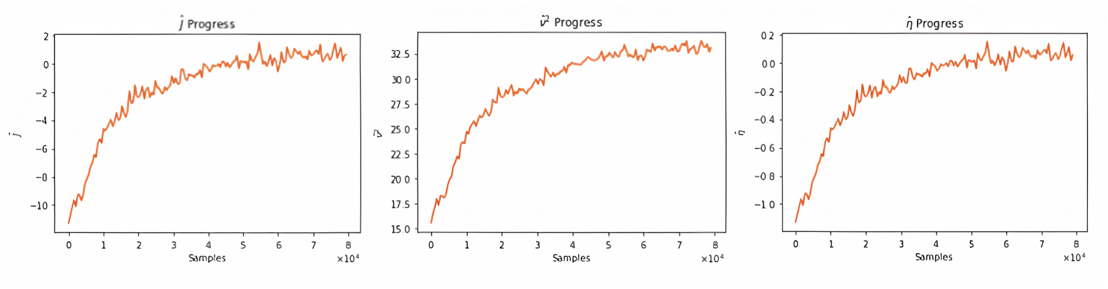

Figure 6.2: Performance of the actor-critic algorithm in the risky chain walk environment at $\lambda = 0$.

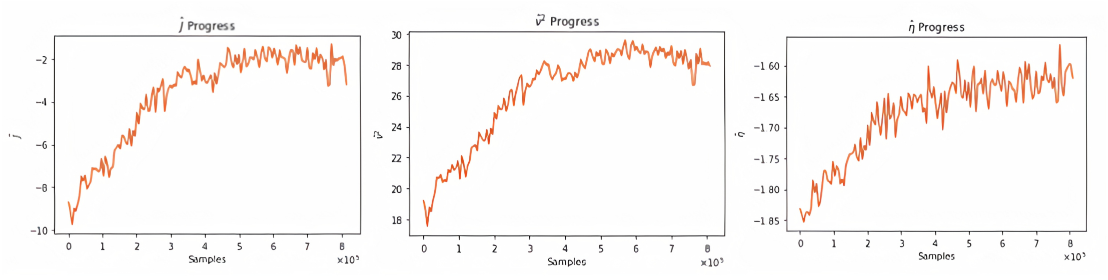

Figure 6.3: Performance of the actor-critic algorithm in the risky chain walk environment at $\lambda = 0.05$.

increase. However, it does not reach the same level of performance of the risk-neutral version, but it achieves lower reward volatility. We will also show how such policies act in the environment in the next section. One can notice from the rightmost figure that the learning progress of the mean volatility is noisier than that of the previous case.

Risk-Averse Objective ($\lambda = 0.07$)

We now increase the value of λ to 0.07, which should increase its sensitivity to risk. Figure 6.4 shows the performance of the algorithm at this level of risk-aversion, again averaged over 10 runs. We can see that at this level of risk-aversion, the behaviour changes dramatically. The algorithm seems to be ready to make big sacrifices on the expected return for the purpose of keeping the volatility low enough. We also see that the learning progress of the mean volatility is even noisier than the previous case.

6.2 Policy Evaluation

We now turn to the policy evaluation problem. Here we fix a policy, and attempt to learn its transformed value function. The nice thing about the risky chain walk environment is that it is easy to obtain the true value function in closed form. This way, we can assess

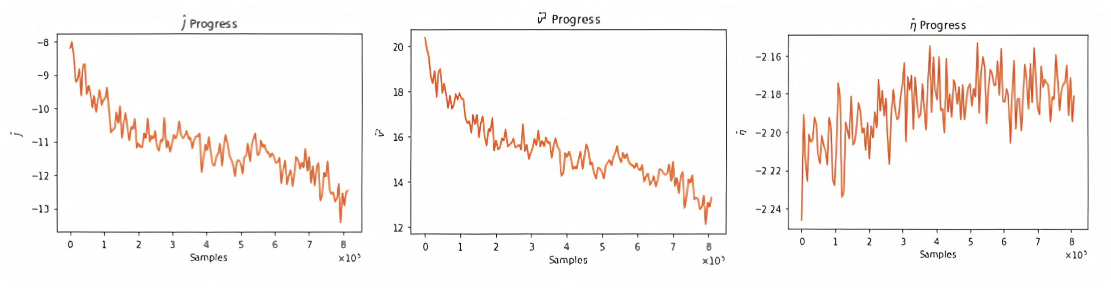

Figure 6.4: Performance of the actor-critic algorithm in the risky chain walk environment at $\lambda = 0.07$.

the learned estimates by comparing them with the true values. For the performance measure, we use the root mean square error between the learned value function and the true one, evaluated according to the stationary distribution of the policy. That is, we consider a performance measure of this form (for the transformed value function):

$$\|V^\lambda - \hat{V}^\lambda\|_{\mu_\pi},$$

where μ_π is the stationary distribution of the policy under evaluation (this kind of norm was defined in (4.2)). In the following, we adopt LSTD as our policy evaluation method, both for the direct method (Algorithm 3.3) and the factored one (Algorithm 4.1). The features we use are also Gaussian radial basis functions like the ones we described in the previous section, but we use 6 means instead of 3. Also in this section, we use independent sampling for each of the involved estimates, expect for V and M which are learned using the same trajectory in Algorithm 4.1. We will consider 3 example policies that were obtained from the training processes of the last section. These policies are not optimal, they are specifically chosen to be somewhat exploratory as to facilitate the presentation of the learning progress. In any case, these policies still convey the expected behaviour under the considered levels of risk-aversion.

Risk-Neutral Case ($\lambda = 0$)

As in the last section, we start with the risk-neutral case. To gain a sense of the behaviour of the selected policy, Figure 6.5 shows its stationary distribution. We can see that the policy spends most of its time at the central region, specifically at the central state, which provides the highest reward.

Since $\lambda = 0$, the transformed value function is the same as the normal one, and our task here is the same as in the risk-neutral policy evaluation case. Figure 6.6 shows the performance, averaged over 30 runs, of LSTD for this problem. The diagram on the left shows the progress of the learned value function. The blue function is the true value function, while the dashed black one is the projection of the true value function to our space of value functions using the norm induced by the stationary distribution. In other words, the dashed black function is the best that we can achieve according to the adopted performance measure. The other functions are the learned ones, which

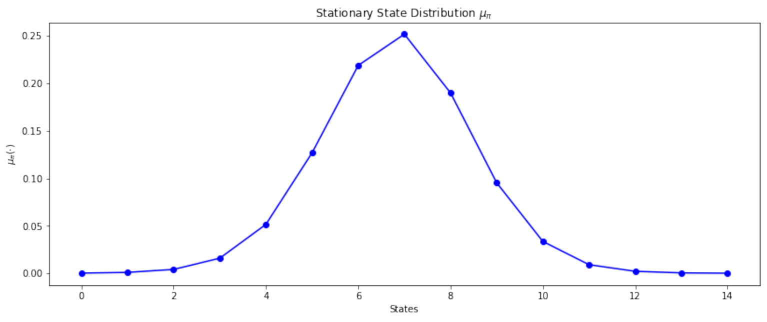

Figure 6.5: The stationary distribution of a risk-neutral policy in the risky chain walk environment.

range from bright green (earliest iterations) to bright red (latest iterations). One can notice that the prediction accuracy at the extreme states is quite poor. However, since these states are seldom visited by the policy, and we are evaluating the prediction error over the stationary distribution, the effect of these states on the performance measure is quite limited, as evidenced by the projected function itself making poor predictions at theses states. The diagram on the right shows how the error is reduced by increasing the length of the (single) trajectory used to learn the value function. The dashed black line represents the error achieved by the projected function.

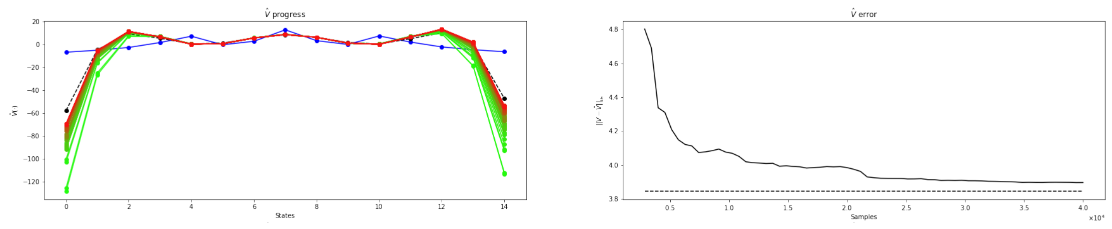

Figure 6.6: The performance of risk-neutral policy evaluation using LSTD in the risky chain walk environment.

Risk-Averse Case ($\lambda = 0.05$)

We now move over to the risk-averse case with $\lambda = 0.05$. We again consider a policy obtained during the training process of the actor-critic algorithm in this setting. Figure 6.7 shows the stationary distribution of the chosen policy. We can see that using this policy, the agent spends the most time in the region around state 10 (which provides a reward of 7) in which the reward variability is less than the central region. With the purpose of estimating the transformed value function, we use Algorithm 4.1, and report its performance, averaged over 20 runs, in Figure 6.8. The reported figure uses the same notation as the one described for Figure 6.6. It reports the progress of the learned estimates compared to the true and the projected versions for the value function, the second moment function, and the transformed value function. Additionally, it shows

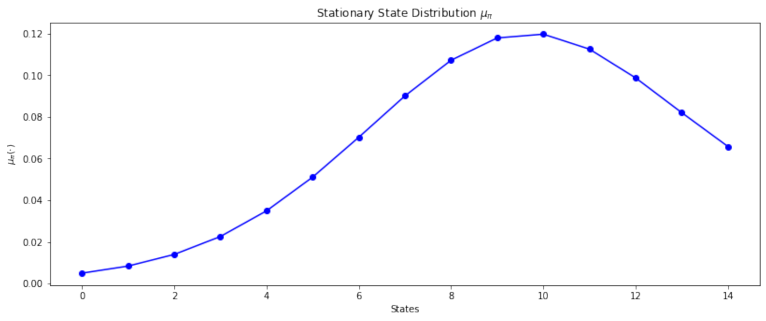

Figure 6.7: The stationary distribution of a risk-averse policy ($\lambda = 0.05$) in the risky chain walk environment.

how the estimation error of each of these functions reduces when the number of samples increases. Similar figures are also reported for the estimation of the expected return. In particular, the figure in the left at the third row shows, in black, the estimates of J over the iterations, and the true value is reported by the horizontal blue line. The figure on its right shows how the error (measured as the absolute value of the difference between the true J and the estimated one) decreases as the number of samples increases. We note here that the length of the trajectories used to estimate J is fixed at 100. Which means that the only way we increase the number of samples for estimating J is by using more trajectories. For completeness, we also report in Figure 6.9 the performance, averaged over 20 runs, when using LSTD to learn the transformed value function using the direct method (i.e. the scheme in Algorithm 3.3), although no significant difference in performance is observed.

Risk-Averse Case ($\lambda = 0.07$)

Finally, we consider the risk-averse case with $\lambda = 0.07$. Considering again a policy obtained during the training process of the actor-critic algorithm at $\lambda = 0.07$, we report its stationary distribution in Figure 6.10. We can see that, as suggested by the findings of the previous section, the policy causes the agent (at the cost of receiving mostly negative rewards) to escape to the extreme ends of the chain, where the reward variability is low. We report in figure 6.11 the performance, averaged over 30 runs, of Algorithm 4.1 when used to estimate the transformed value function of the policy in question.

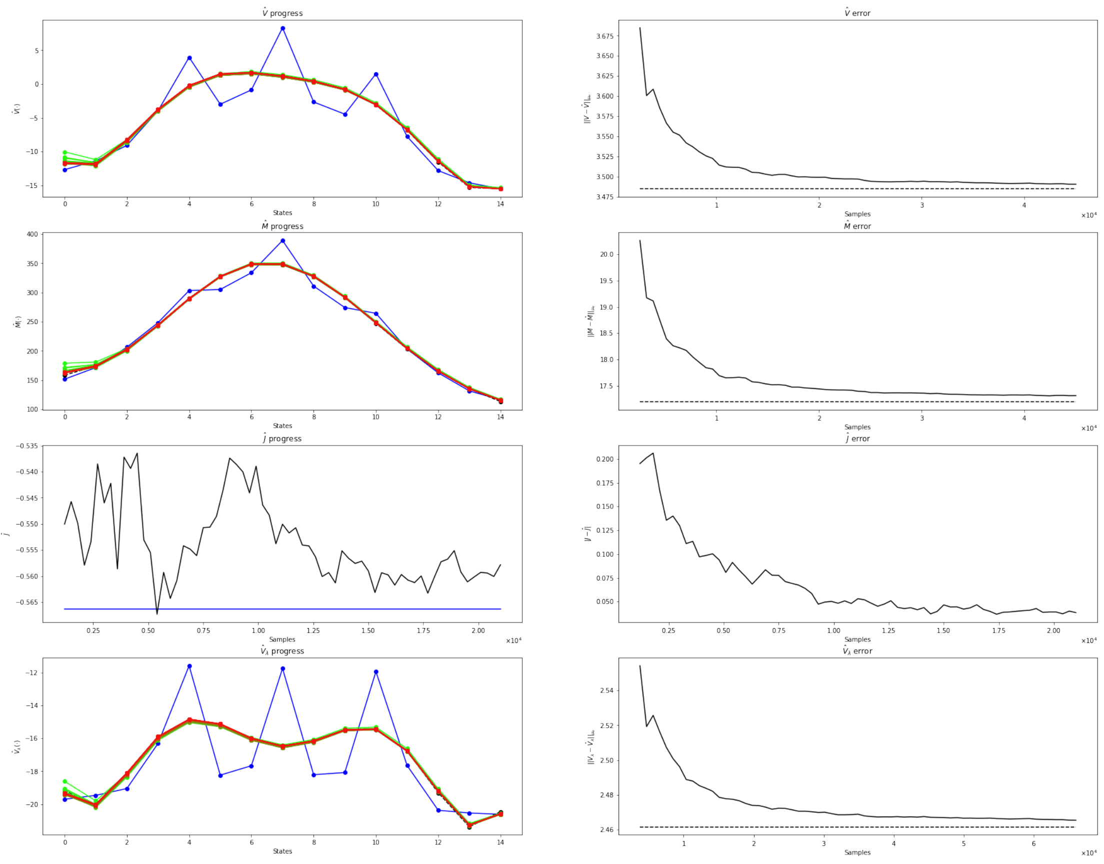

Figure 6.8: The performance of risk-averse ($\lambda = 0.05$) policy evaluation using Algorithm 4.1 in the risky chain walk environment.

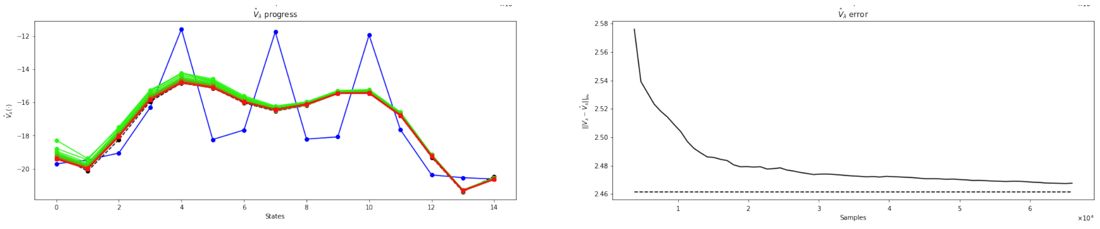

Figure 6.9: The performance of risk-averse ($\lambda = 0.05$) policy evaluation using LSTD with the direct method in the risky chain walk environment.

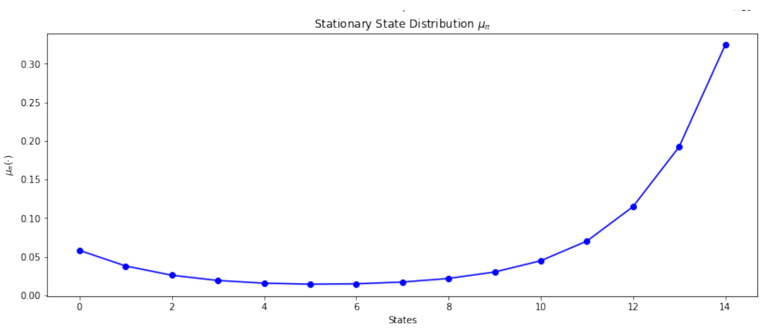

Figure 6.10: The stationary distribution of a risk-averse policy ($\lambda = 0.07$) in the risky chain walk environment.

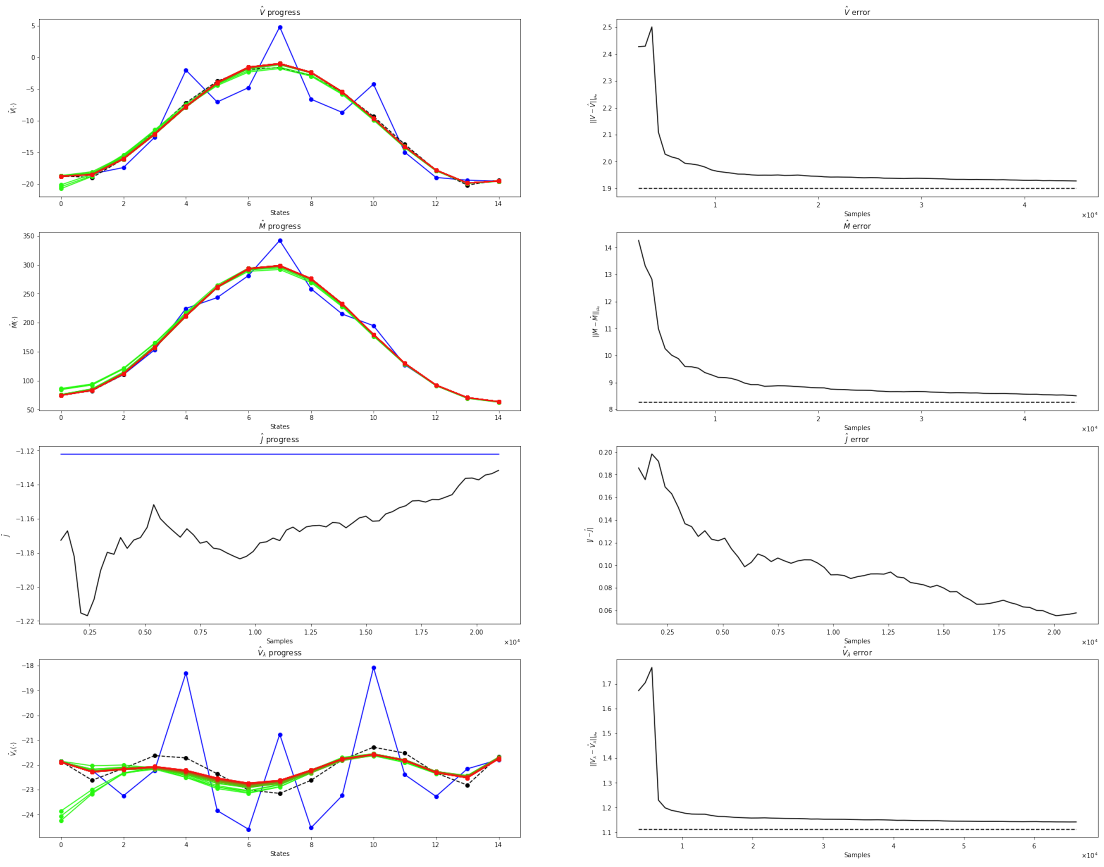

Figure 6.11: The performance of risk-averse ($\lambda = 0.07$) policy evaluation using Algorithm 4.1 in the risky chain walk environment.

Conclusions

In this thesis, the focus was on analyzing the sample complexity of some risk-averse reinforcement learning algorithms. In particular, we adopted the mean volatility [7] objective, through which we aim to achieve an adequate balance between maximizing the standard objective of the expected return and minimizing the reward volatility, which is the variance of the per-step reward. We considered two different methods for policy evaluation when adopting the mean volatility objective (i.e. for estimating the transformed value function): the factored method (Algorithm 3.4) and the direct method (Algorithm 3.3), both of which involve an initial step where we estimate the expected return of the policy under evaluation (Algorithm 3.1). While the direct method uses this estimate of the expected return to perform a reward transformation and proceeds as in the risk-neutral setting, the factored method relies on factorizing the transformed value function into a number of terms (Namely: the value function V, the second moment value function M, and the expected return J) that are estimated separately, without the need to involve the estimated expected return in any reward transformation. We have seen in Chapter 4 that the factored method allowed us to derive a general template for bounding the error on the estimated transformed value function using the error bounds on the estimates of the terms in its factorization. This allowed us to study the effect of the estimation error of the expected return (and its square) on the accuracy of the transformed value function estimate, regardless of the methods we used to estimate V and M. We have then exemplified the use of this template by providing a full finite-time bound for the factored mean-volatility LSTD algorithm (Algorithm 4.1). A similar general template is not provided for the direct method since the effect of the error in the estimate of the expected return on the accuracy of the estimated transformed value function depends on the adopted risk-neutral policy evaluation method. We then provided finite-sample analysis for a full actor-critic algorithm (Algorithm 5.1) for optimizing the mean volatility, where the critic (Algorithm 5.2) used the direct method to estimate the transformed value function. Finally, we tested the proposed algorithms in a simple chain walk environment to assess their soundness and performance.

One of the main goals of the analysis of any of the mentioned algorithms was to understand how their sample complexity compares to their risk-neutral counterparts. In particular, we have seen in the algorithms we analyzed that when the expected return is estimated using its own batch of data, the sample complexity of the algorithm compared to its risk-neutral counterpart is worsened only if the sample complexity (in the relevant sense) of the estimation of the expected reward and its square is worse than that of the risk-neutral algorithm. This indeed was the case for the factored mean-volatility LSTD algorithm, while we were able to achieve the same complexity for the actor-critic algorithm. We have also focused on reusing samples when possible, and understanding the effect of doing so. In particular, our analysis of the factored method did not prohibit us from estimating multiple terms using the same data.

Future Work

There are multiple directions for future work, some of which we summarize below:

- One direction is to understand the effect of estimating the expected return and the transformed value function in the direct method using the same data. For example, we can study the effect of doing so on our actor-critic algorithm. Although, we have shown that the complexity is not worsened, we still need an extra batch of samples for estimating J. Thus, it would be interesting to understand the effect reusing these samples for training the critic or the actor.
- Another direction is to extend the analysis to more advanced methods. In particular, our use of a simple linear TD critic and a plain actor-critic approach limits the applicability of the algorithm. Possible extensions include using the natural actor-critic approach or using more advanced critics that use, for example, neural networks or regression trees.
- Off-policy approaches are particularly appealing for real-life applications since they can improve sample efficiency and can limit the exposure of the agent to the environment during the training phase when a simulated environment is not available. Thus, extending our analysis to the off-policy setting is another interesting direction.
- Lastly, we can also consider extending the analysis to other risk measures like the semi-deviation or the CVaR, which are usually more suitable choices than the variance as we have discussed before.

Bibliography

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[4] Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *J. Artif. Int. Res.*, 24(1):81–108, July 2005.

[5] Aviv Tamar, Dotan Di Castro, and Shie Mannor. Learning the variance of the reward-to-go. *Journal of Machine Learning Research*, 17(13):1–36, 2016.

[6] Aviv Tamar, Yonatan Glassner, and Shie Mannor. Optimizing the cvar via sampling, 2014.

[7] Lorenzo Bisi, Luca Sabbioni, Edoardo Vittori, Matteo Papini, and Marcello Restelli. Risk-averse trust region optimization for reward-volatility reduction. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4583–4589. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Special Track on AI in FinTech.

[8] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, Belmont, MA, 1996.

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

[10] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *J. Mach. Learn. Res.*, 4:1107–1149, December 2003.

[11] Tengyu Xu, Zhe Wang, and Yingbin Liang. Improving sample complexity bounds for (natural) actor-critic algorithms. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 4358–4369. Curran Associates, Inc., 2020.

[12] Jalaj Bhandari, Daniel Russo, and Raghav Singal. A finite time analysis of temporal difference learning with linear function approximation. *Oper. Res.*, 69:950–973, 2018.

[13] Justin A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, Nov 2002.

[14] Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, Mar 1996.

[15] Ronald Williams and Leemon C. Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, Northeastern University, 1993.

[16] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 30–37. Morgan Kaufmann, San Francisco (CA), 1995.

[17] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(18):503–556, 2005.

[18] Philip Thomas. Bias in natural actor-critic algorithms. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 441–448, Bejing, China, 22–24 Jun 2014. PMLR.

[19] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000.

[20] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.

[21] Shun-ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, 02 1998.

[22] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Natural actor-critic. In João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo, editors, *Machine Learning: ECML 2005*, pages 280–291, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[23] M. P. Deisenroth, G. Neumann, and J. Peters. A survey on policy search for robotics, 2013.

[24] Francisco S. Melo and Manuel Lopes. Fitted natural actor-critic: A new algorithm for continuous state-action mdps. In Walter Daelemans, Bart Goethals, and Katha-

rina Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 66–81, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[25] Javier García, Fern, and o Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(42):1437–1480, 2015.

[26] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, page 267–274, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[27] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

[28] Clement Gehring and Doina Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '13, page 1037–1044, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

[29] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczynski. *Lectures on Stochastic Programming: Modeling and Theory, Second Edition*. Society for Industrial and Applied Mathematics, USA, 2014.

[30] Yingjie Fei, Zhuoran Yang, Yudong Chen, Zhaoran Wang, and Qiaomin Xie. Risk-sensitive reinforcement learning: Near-optimal risk-sample tradeoff in regret. In *NeurIPS*, 2020.

[31] Matthew J. Sobel. The variance of discounted markov decision processes. *Journal of Applied Probability*, 19(4):794–802, 1982.

[32] Aviv Tamar, Dotan Di Castro, and Shie Mannor. Policy gradients with variance related risk criteria. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1651–1658, Madison, WI, USA, 2012. Omnipress.

[33] Shie Mannor and John N. Tsitsiklis. Mean-variance optimization in markov decision processes. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 177–184, Madison, WI, USA, 2011. Omnipress.

[34] Aviv Tamar. Risk-sensitive and efficient reinforcement learning algorithms, 2015.

[35] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical Finance*, 9(3):203–228, 1999.

[36] Aviv Tamar, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor. Policy gradient for coherent risk measures. In C. Cortes, N. Lawrence, D. Lee,

M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[37] Shangtong Zhang, Bo Liu, and Shimon Whiteson. Mean-variance policy iteration for risk-averse reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):10905–10913, May 2021.

[38] Alessandro Lazaric, Mohammad Ghavamzadeh, and Rémi Munos. Finite-sample analysis of least-squares policy iteration. *Journal of Machine Learning Research*, 13:3041–3074, 2012.

[39] R. Sharma, R. Kumar, R. Saini, and G. Kapoor. Complementary upper bounds for fourth central moment with extensions and applications, 2015.

[40] Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to probability.* Optimization and computation series. Athena scientific, Belmont, 2nd ed edition, 2008.

[41] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[42] Jordanka Angelova. On moments of sample mean and variance. *International Journal of Pure and Applied Mathematics*, 79, 01 2012.

[43] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, third edition, 1996.

[44] Michael Grant (https://scicomp.stackexchange.com/users/3907/michael grant). How to prove the 2-norm of an invertible matrix is exactly the reciprocal of its minimum singular value? Computational Science Stack Exchange. URL:https://scicomp.stackexchange.com/q/10465 (version: 2014-01-06).

[45] M. Dahleh, Munther A. Dahleh, and G. Verghese. Lectures on dynamic systems and control, 2004.

Appendix A

Auxiliary Lemmas

Lemma A.1. *Suppose A is an $n \times n$ invertible matrix, then*

$$\|A^{-1}\|_2 = \frac{1}{\min_i \sigma_i},$$

where, for a matrix, $\|\cdot\|_2$ denotes its spectral norm, and σ_i is the i^{th} singular value of A.

Proof. (The following proof is due to [44].) By Theorem 4.3 in [45], we have that

$$\min_i \sigma_i = \inf_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}.$$

And since A is invertible, $\min_i \sigma_i > 0$. We then have that

$$\begin{aligned}
\frac{1}{\min_i \sigma_i} &= \sup_{x \neq 0} \frac{\|x\|_2}{\|Ax\|_2} \\
&= \sup_{A^{-1}y \neq 0} \frac{\|A^{-1}y\|_2}{\|y\|_2} \\
&= \sup_{y \neq 0} \frac{\|A^{-1}y\|_2}{\|y\|_2} \\
&= \|A^{-1}\|_2,
\end{aligned}$$

where we have made the substitution $Ax = y$ and utilized the fact that $A^{-1}y = 0$ iff $y = 0$ since A is invertible. $\square$

Lemma A.2. *Consider d real valued vectors $a_1, \ldots, a_d \in \mathbb{R}^n$, we have that:*

(i) $\forall i, j \in \{1, \ldots, d\} : \ |\langle a_i, b_j \rangle| \leq \frac{1}{2} \left(\|a_i\|_2^2 + \|b_i\|_2^2 \right).$

(ii) $\left\| \sum_{i=1}^{d} a_i \right\|_2^2 \leq d \sum_{i=1}^{d} \|a_i\|_2^2$

Proof. For any i, j we have:

$$\|a_i + a_j\|_2^2 = \|a_i\|_2^2 + \|a_j\|_2^2 + 2\langle a_i, a_j\rangle \geq 0, \text{ and } \|a_i - a_j\|_2^2 = \|a_i\|_2^2 + \|a_j\|_2^2 - 2\langle a_i, a_j\rangle \geq 0,$$

hence, trivially:

$$-\frac{1}{2}\|a_i\|_2^2 - \frac{1}{2}\|a_j\|_2^2 \leq \langle a_i, a_j\rangle, \text{ and } \frac{1}{2}\|a_i\|_2^2 + \frac{1}{2}\|a_j\|_2^2 \geq \langle a_i, a_j\rangle,$$

which proves **(i)**.

By repeatedly applying **(i)** to the cross-terms of $\left\|\sum_{i=1}^d a_i\right\|_2^2$, we obtain:

$$\left\|\sum_{i=1}^d a_i\right\|_2^2 = \sum_{i=1}^d \|a_i\|_2^2 + 2\sum_{i>j}\langle a_i, a_j\rangle \leq \sum_{i=1}^d \|a_i\|_2^2 + \sum_{i>j}\Big(\|a_i\|_2^2 + \|a_j\|_2^2\Big) = d\sum_{i=1}^d \|a_i\|_2^2,$$

since each index is counted $d-1$ times in the summation. $\square$

Appendix B

Smoothness Proofs

Lemma B.1. *Suppose Assumptions 5.1 and 5.4 hold, then* $\forall\, \theta_1, \theta_2 \in \mathbb{R}^{d_\theta}$, *we have*

$$\|d_{I,\theta_1}(.,.) - d_{I,\theta_2}(.,.)\|_{TV} \leq C_d \|\theta_1 - \theta_2\|_2,$$

where $C_d := C_\pi \left(1 + \lceil log_\rho \kappa^{-1} \rceil + \frac{1}{1-\rho}\right)$, *and* $d_{I,\theta}(s,a) := d_{I,\theta}(s)\pi(a|s)$, *where* $d_{I,\theta}(\cdot)$ *is the (normalized) discounted state distribution when using policy* π_θ *and starting from* $I(\cdot)$, *which is an initialization distribution over the states; it can be taken as* $\mu_0(.)$ *(the initial state distribution) or* $P(.|s', a')$ *for any fixed state-action pair* (s', a').

Proof. See Lemma 3 in [11]. □

Lemma B.2. *Suppose Assumptions 5.1 and 5.4 hold, then* $\forall\, \theta_1, \theta_2 \in \mathbb{R}^{d_\theta}$, *we have*

$$|J_{\theta_1} - J_{\theta_2}| \leq L_J \|\theta_1 - \theta_2\|_2,$$

where $L_J := 2R_{\max}(C_d + C_\pi)$.

Proof.

$$\begin{aligned}
&|J_{\theta_1} - J_{\theta_2}| \\
&= \left|(1-\gamma)\int_s (V_{\theta_1}(s) - V_{\theta_2}(s))\mu(ds)\right| \\
&\leq (1-\gamma)\int_s |V_{\theta_1}(s) - V_{\theta_2}(s)|\mu(ds) \\
&\leq (1-\gamma)\int_s \left|\int_a Q_{\theta_1}(a,s)\pi_{\theta_1}(da|s) - \int_a Q_{\theta_2}(a,s)\pi_{\theta_2}(da|s)\right|\mu(ds) \\
&\leq (1-\gamma)\int_s \left|\int_a Q_{\theta_1}(a,s)\pi_{\theta_1}(da|s) \pm \int_a Q_{\theta_2}(a,s)\pi_{\theta_1}(da|s) - \int_a Q_{\theta_2}(a,s)\pi_{\theta_2}(da|s)\right|\mu(ds) \\
&\leq (1-\gamma)\int_s \int_a |(Q_{\theta_1}(a,s) - Q_{\theta_2}(a,s))|\pi_{\theta_1}(da|s)\mu(ds)
\end{aligned}$$

$$+ (1-\gamma)\int_s\int_a |Q_{\theta_2}(a,s)|\,|\pi_{\theta_1}(da|s) - \pi_{\theta_2}(da|s)|\mu(ds)$$

By Lemma 4 in [11], $|Q_{\theta_1}(s,a) - Q_{\theta_2}(s,a)| \le \frac{2R_{\max}C_d}{1-\gamma}\|\theta_1-\theta_2\|_2 \; \forall\,(s,a)\in S\times A$. Using this, and assumption 5.1.v, we have that

$$\begin{aligned}
&|J_{\theta_1} - J_{\theta_2}| \\
&\le 2R_{\max}C_d\|\theta_1-\theta_2\|_2 + R_{\max}\int_s\int_a |\pi_{\theta_1}(da|s) - \pi_{\theta_2}(da|s)|\mu(ds) \\
&\le 2R_{\max}C_d\|\theta_1-\theta_2\|_2 + 2R_{\max}C_\pi\|\theta_1-\theta_2\|_2 \\
&= 2R_{\max}(C_d + C_\pi)\|\theta_1-\theta_2\|_2
\end{aligned}$$

□

Lemma B.3. *Suppose Assumptions 5.1 and 5.4 hold, then* $\forall\,\theta_1,\theta_2\in\mathbb{R}^{d_\theta}$ *and* $\forall\,(s,a)\in S\times A$*, we have*

$$\left|Q^\lambda_{\theta_1}(s,a) - Q^\lambda_{\theta_2}(s,a)\right| \le L_{Q^\lambda}\|\theta_1-\theta_2\|_2,$$

where $L_{Q^\lambda} := \frac{2C_dR_{\lambda,\max}+4\lambda L_J R_{\max}}{1-\gamma} = \frac{2C_dR_{\max}+8\lambda R^2_{\max}(2C_d+C_\pi)}{1-\gamma}$*, and* $\lambda \ge 0$.

Proof. By definition,

$$\begin{aligned}
Q^\lambda_\theta(s,a) &= \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s'\sim d_\theta(\cdot|s,a)\\ a'\sim\pi_\theta(\cdot|s')}}\left[R^\lambda_\theta(s,a)\right] \\
&= \frac{1}{1-\gamma}\int_{s'}\int_{a'} R^\lambda_\theta(s',a')d_\theta(ds'|s,a)\pi_\theta(da'|s') \\
&= \frac{1}{1-\gamma}\int_{(s',a')} R^\lambda_\theta(s',a')d_\theta(ds',da'|s,a),
\end{aligned}$$

where $d_\theta(s',a'|s,a) := d_\theta(s'|s,a)\pi_\theta(a'|s')$, and $d_\theta(\cdot|s,a)$ is the (normalized) discounted state distribution when using policy π_θ after taking action a in state s. We then have that

$$\begin{aligned}
&(1-\gamma)\left|Q^\lambda_{\theta_1}(s,a) - Q^\lambda_{\theta_2}(s,a)\right| \\
&= \left|\int_{(s',a')}\left[R^\lambda_{\theta_1}(s',a')d_{\theta_1}(ds',da'|s,a) - R^\lambda_{\theta_2}(s',a')d_{\theta_2}(ds',da'|s,a)\right]\right| \\
&= \left|\int_{(s',a')}\left[R^\lambda_{\theta_1}(s',a')d_{\theta_1}(ds',da'|s,a) \pm R^\lambda_{\theta_1}(s',a')d_{\theta_2}(ds',da'|s,a) - R^\lambda_{\theta_2}(s',a')d_{\theta_2}(ds',da'|s,a)\right]\right| \\
&= \left|\int_{(s',a')} R^\lambda_{\theta_1}(s',a')(d_{\theta_1}(ds',da'|s,a) - d_{\theta_2}(ds',da'|s,a))\right| \\
&\quad + \left|\int_{(s',a')} (R^\lambda_{\theta_1}(s',a') - R^\lambda_{\theta_2}(s',a'))d_{\theta_2}(ds',da'|s,a)\right|
\end{aligned}$$

$$
\begin{aligned}
&\leq \int_{(s',a')} \left|R^\lambda_{\theta_1}(s',a')\right| \left|d_{\theta_1}(ds',da'|s,a) - d_{\theta_2}(ds',da'|s,a)\right| \\
&\quad + \int_{(s',a')} \left|R^\lambda_{\theta_1}(s',a') - R^\lambda_{\theta_2}(s',a')\right| d_{\theta_2}(ds',da'|s,a) \\
&\leq R_{\lambda,\max} \int_{(s',a')} \left|d_{\theta_1}(ds',da'|s,a) - d_{\theta_2}(ds',da'|s,a)\right| \\
&\quad + \int_{(s',a')} \left|2\lambda R(s',a')(J_{\theta_1} - J_{\theta_2}) - \lambda(J^2_{\theta_1} - J^2_{\theta_2})\right| d_{\theta_2}(ds',da'|s,a) \\
&\leq 2C_d R_{\lambda,\max}\|\theta_1 - \theta_2\|_2 \\
&\quad + \int_{(s',a')} \left|2\lambda R(s',a')(J_{\theta_1} - J_{\theta_2}) - \lambda(J_{\theta_1} + J_{\theta_2})(J_{\theta_1} - J_{\theta_2})\right| d_{\theta_2}(ds',da'|s,a) \\
&\leq 2C_d R_{\lambda,\max}\|\theta_1 - \theta_2\|_2 \\
&\quad + \int_{(s',a')} \left|\lambda(2R(s',a') - (J_{\theta_1} + J_{\theta_2}))\right| \left|J_{\theta_1} - J_{\theta_2}\right| d_{\theta_2}(ds',da'|s,a) \\
&\leq 2C_d R_{\lambda,\max}\|\theta_1 - \theta_2\|_2 + 4\lambda R_{\max}|J_{\theta_1} - J_{\theta_2}| \\
&\leq 2C_d R_{\lambda,\max}\|\theta_1 - \theta_2\|_2 + 4\lambda L_J R_{\max}\|\theta_1 - \theta_2\|_2 \\
&= (2C_d R_{\lambda,\max} + 4\lambda L_J R_{\max})\,\|\theta_1 - \theta_2\|_2 \\
&= (2C_d R_{\max} + 8\lambda R^2_{\max}(2C_d + C_\pi))\,\|\theta_1 - \theta_2\|_2.
\end{aligned}
$$

□

Lemma B.4. *Suppose Assumptions 5.1 and 5.4 hold, then $\forall\, \theta_1, \theta_2$, we have*

$$\|\nabla\eta_{\theta_1} - \nabla\eta_{\theta_2}\|_2 \leq L_\eta\|\theta_1 - \theta_2\|_2,$$

where $L_\eta := \frac{2R_{\lambda,\max}C_\psi C_d}{1-\gamma} + C_\psi L_{Q^\lambda} + \frac{R_{\lambda,\max}L_\psi}{1-\gamma}$, and $\lambda \geq 0$.

Proof.

$$
\begin{aligned}
&\|\nabla\eta_{\theta_1} - \nabla\eta_{\theta_2}\|_2 \\
&= \left\| \int_{(s,a)} \left[\psi_{\theta_1}(s,a)Q^\lambda_{\theta_1}(s,a)d_{\mu,\theta_1}(ds,da) - \psi_{\theta_2}(s,a)Q^\lambda_{\theta_2}(s,a)d_{\mu,\theta_2}(ds,da)\right] \right\|_2 \\
&\leq \int_{(s,a)} \left\|Q^\lambda_{\theta_1}(s,a)\psi_{\theta_1}(s,a)\right\|_2 |d_{\mu,\theta_1}(ds,da) - d_{\mu,\theta_2}(ds,da)| \\
&\quad + \int_{(s,a)} \left|Q^\lambda_{\theta_1}(s,a) - Q^\lambda_{\theta_2}(s,a)\right| \|\psi_{\theta_1}(s,a)\|_2\, d_{\mu,\theta_2}(ds,da) \\
&\quad + \int_{(s,a)} \left|Q^\lambda_{\theta_2}(s,a)\right| \|\psi_{\theta_1}(s,a) - \psi_{\theta_2}(s,a)\|_2\, d_{\mu,\theta_2}(ds,da) \\
&\leq \frac{2R_{\lambda,\max}C_\psi C_d}{1-\gamma}\|\theta_1 - \theta_2\|_2 + C_\psi L_{Q^\lambda}\|\theta_1 - \theta_2\|_2 + \frac{R_{\lambda,\max}L_\psi}{1-\gamma}\|\theta_1 - \theta_2\|_2 \\
&= \left(\frac{2R_{\lambda,\max}C_\psi C_d}{1-\gamma} + C_\psi L_{Q^\lambda} + \frac{R_{\lambda,\max}L_\psi}{1-\gamma}\right)\|\theta_1 - \theta_2\|_2,
\end{aligned}
$$

where the last inequality follows from Assumption 5.1, Lemma B.1, and Lemma B.3. $\square$