Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Doctoral Programme In Information Technology

Safe Policy Optimization

Doctoral Dissertation of:
Matteo Papini

Supervisor:
Prof. Marcello Restelli

2020 – Cycle XXXIII

Abstract

Policy Optimization (PO) is a family of reinforcement learning algorithms that is particularly suited to real-world control tasks due to its ability of managing high-dimensional decision variables and noisy signals. This also makes PO one of the most pressing targets of safety concerns. Outside of simulation, the trial-and-error behavior typical of learning agents can have concrete, potentially catastrophic consequences. The design of reliable adaptive agents for real-world settings requires, first of all, a better theoretical understanding of the learning algorithms used to train them. In this dissertation, we highlight the potential and limitations of existing policy optimization techniques, with a special focus on policy gradient algorithms. We study theoretical properties of policy gradients that are relevant to safety. We establish novel guarantees of monotonic performance improvement and convergence. We also study the trade-offs that safety requirements inevitably engage with sample complexity and exploration. Besides improving the theoretical understanding of policy gradient methods, we design new algorithms with more desirable properties, and evaluate them on simulated continuous control tasks.

"Don't go on multiplying the mysteries," he said. "They should be kept simple. Bear in mind Poe's purloined letter, bear in mind Zangwill's locked room."
"Or made complex," replied Dunraven. "Bear in mind the universe."

— Jorge Luis Borges, *Abenjacán el Bojarí, Dead in his Labyrinth*

Acknowledgments

First and foremost, this work would not exist without the phenomenal mentorship of Prof. Marcello Restelli.

Research is a team effort. All the original contributions I present in this dissertation were developed in joint work with remarkable co-authors—in addition to Marcello.

Smoothing Policies and Safe Policy Gradients (Papini et al., 2019b), discussed in Chapter 5, is a joint work with Matteo Pirotta, and a follow-up on his previous work on policy gradients. Matteo deserves additional thanks as unofficial mentor, since I have often been following in his footsteps.

Policy Optimization with Importance Sampling (Metelli et al., 2018, 2020b), presented in Chapter 6, is a joint work with Alberto Maria Metelli, Francesco Faccio, and Nico Montali.

Stochastic Variance-Reduced Policy Gradient (Papini et al., 2018), presented in Chapter 7, is a joint work with Damiano Binaghi, Giuseppe Canonaco, and Matteo Pirotta.

Chapter 8 presents work done with Andrea Battistello (Papini et al., 2020).

Research is a conversation. My views and knowledge have been shaped in endless ways by working, discussing, brainstorming with amazing people, from my desk to far-away lands—too many to name. I shall only mention my office mates Alberto, Andrea and Giorgia for the constant exchange of ideas, the healthy competition, and inspiration of a kind that can only come from equals.

Seeing this conversation continue through my written works was one of the greatest satisfactions of my PhD. I must thank Pan Xu and colleagues for letting me include their results (Xu et al., 2020) on variance-reduced policy gradients (Figure 7.3), and for providing additional information on their experiments. I also decided to report in full detail their theoretical analysis (Xu et al., 2019) of SVRPG (Papini et al., 2018) in Chapter 7, since it greatly helps demonstrating the significance of our algorithm. All exogenous contributions are fully documented.

I also thank the reviewers for their invaluable feedback: Gergely Neu, Matthieu Geist and Gerhard Neumann. Minor changes were made to this final version to address their suggestions and to update some references.

Research is a life choice. I wholeheartedly thank my friends and family for supporting me during my journey on this uneven road.

The truth is, research is fun! For this, I thank my colleagues at AIRLab.

Matteo Papini
Milano
February 19, 2021

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

A-POIS Action–based POIS 117

AC Actor Critic 54

BE Bellman Error 23

BH Balance Heuristic 110

CEM Cross Entropy Method 64

CMDP Constrained Markov Decision Process 72

D-POIS per-Decision action-based POIS 124

DDPG Deep Deterministic Policy Gradient 59

DPG Deterministic Policy Gradient 59

DQN Deep Q-Network 27

EM Expectation Maximization 65

ES Evolution Strategies 64

FIM Fisher Information Matrix 33

FO First-order Oracle 90

FSO First-Order Stochastic Oracle 91

GAE Generalized Advantage Estimation 56

GD full Gradient Descent 138

IS Importance Sampling 25

KL Kullback-Leibler divergence 34

LQR Linear Quadratic Regulator 103

MC Monte Carlo 23

List of Symbols

$[n]$	Set $\{1, 2, \ldots, n\}$
Δ_Ω	Set of probability measures over measurable space Ω
$\mathcal{N}$	Gaussian distribution
$\mathcal{U}$	Uniform distribution
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
p	Transition kernel
r	Reward function
γ	Discount factor
μ	Starting-state distribution
π	Policy
T^π	Bellman's expectation operator for policy π
T^*	Bellman's optimality operator
V^π	Value function of policy π
Q^π	Quality function of policy π
d_μ^π	State-occupancy distribution of policy π
τ	Trajectory
H	Task horizon
$\boldsymbol{\theta}$	Vector of policy parameters
$\nabla_{\boldsymbol{\theta}}$	Gradient w.r.t. $\boldsymbol{\theta}$
$\nabla^2_{\boldsymbol{\theta}}$	Hessian w.r.t. $\boldsymbol{\theta}$

Introduction 1

Reinforcement Learning (RL Sutton and Barto, 2018) is currently the most promising approach to decision making for autonomous agents. Its fundamental principle is biologically inspired, simple and as general as can be: let the agent interact with the unknown environment, provide it with a reward signal informing it of the appropriateness of its actions and let it autonomously learn to maximize rewards. This makes *Reinforcement Learning* (RL) applicable to any sequential decision problem, from selecting the advertisements to display on a website (Thomas et al., 2015b) to controlling humanoid robots (Peters et al., 2003), regardless of uncertainty or lack of prior domain knowledge...at least in principle.

Indeed, RL achieved marvelous results in games, from the millenary Go board game (Silver et al., 2017) to modern multi-player videogames (Berner et al., 2019), creating competitive adversaries for human world champions. More than on novel algorithmic insights, these recent successes depend on the great representational power of deep neural networks (Goodfellow et al., 2016) paired with monstrous computational power. The unprecedented availability of data and compute is what fueled the success of machine learning in general over the last decade. In reinforcement learning, the two tend to coincide, since the data RL agents need are interactions with the environment.[1] In applications like games, where the environment is easy to simulate (or it is itself a simulation, like in videogames) gathering interaction data is just a matter of computational power.

Motivation: RL for real life. Moving from games to real-world applications, new challenges arise. We use robotic control as an ongoing example, but the same could be said of software agents that must interact with humans (like recommender systems on the web) or take decisions that affect physical entities (e.g., helping to select the treatment for hospitalized patients, Thapa et al., 2005).

[1] Learning from historical interaction data is possible but more challenging. See for instance Section 4.5.1.

First, collecting data can be a slow (since the agent is subject to real time) and expensive (since physical systems can consume a lot of energy and wear out) process. This typically dominates the time and expense required for actual learning, which is a purely computational process. Simulated experience can alleviate this problem, but engineering a reliable simulation is simply not possible in many applications. This makes *sample complexity*, the amount of data that is required for learning, a fundamental issue, if not the most important one for the successful application of RL to real-world problems.

The second challenge is a universal dilemma of sequential decision making, but is even more relevant is real-life RL: *exploration*. Learning from scratch, an agent needs to perform some actions just to increase its knowledge of the world. This exploratory behavior may be suboptimal according to its current knowledge, but unlock the ability to learn even better behavior. In a simulated environment, we typically only care about the final solution, so exploratory behavior is almost always worth the effort. Instead, when the agent is learning in the real world, intermediate performance can really matter. Exploratory behavior that sacrifices immediate performance must ensure this is repaid by future improvements. We may call this the *cost of exploration*, to which some users (e.g., a bank using RL to decide how to invest its money, or a factory owner that bought a reinforcement-learning robot to increase production) could be particularly sensitive. This adds to the already difficult problem of designing effective exploration strategies for learning agents.

The third great challenge, and possibly the most urgent one, is *safety*. Learning in the real world can have concrete consequences, ranging from monetary losses to irrecoverable damage, up to threatening the life of human operators. Hence, it is important to identify all the potential hazards, and to reduce risk as much as possible. Careful engineering of the agent's hardware and of the surrounding environment can help a lot by reducing the number of hazards (Büchler et al., 2020). However, it is equally important to design learning algorithms that do not take unnecessary or plain unacceptable risks. The very problem of defining safety is not easy, and has produced a wide range of diverse approaches (García and Fernández, 2015; Amodei et al., 2016). In fact, designing a truly safe RL agent requires to intervene at multiple levels: in the definition of reward (to ensure it is aligned with our desires), in the planning of interaction, and in the design of the learning procedure itself.

Safety concerns. If we wish to apply RL to real-world problems, we need to explicitly face these three challenges[2]. To blindly apply the algorithms that worked so well for games would be a very literal and serious instance of the *ludic fallacy* (Taleb, 2007). This dissertation is specifically on the problem of safety. However, safety is so entangled with the other two challenges that it can hardly be studied in isolation. More samples can help the agent build a better understanding of the world and the effects of his actions, which is fundamental to reduce risk. This creates a first

[2]We could add a fourth one, that we have included under the concept of safety: agent alignment. While in games the objective is typically clear, in real-world scenarios it may be challenging to encode the needs of the user into a monolithic concept of goal, or a scalar reward signal.

tradeoff between safety and sample complexity. On the other hand, spending a lot of time collecting data under suboptimal behavior can itself be risky. Quickly converging to an optimal solution could actually be safer on the long run. Similarly, exploratory behavior can be dangerous, especially if it involves random actions. However, the knowledge it provides can help achieve safer policies on the long run. The triangular relation is completed by noting that exploration requires many data, but can speed-up the learning process overall. These fundamental problems of real-world RL are all tied together, and their relative importance really depends on how we define safety and on the time horizon we care to consider.

In this manuscript we take the so-called Seldonian approach to RL safety (Thomas et al., 2019). This means we do not focus on how safety concerns can be embedded in the problem definition, nor on how the agent can plan its interactions to reduce hazards, but on the risks introduced by the learning process itself. In other words, we focus on preventing the negative effects of *epistemic uncertainty*, the lack of knowledge of the environment and its workings that can lead the agent to *learn* unsafe behavior. This is a general issue of machine learning systems, with the difference that in real-world RL we also care about the safety of intermediate solutions, since they shape the concrete agent-environment interaction. In fact, our concept of safety is particularly related to the problem of controlling the cost of exploration. The responsibility of ensuring safety in this sense is entirely of the designer of the reinforcement learning algorithm. Since the responsibility of encoding specific safety requirements is left to the user, this should be intended as a complement, not as a substitution, of other safe RL approaches. It requires a deep understanding, and often a modification, of RL algorithms developed without the safety problem in mind.

Policy optimization. This dissertation addresses the safety challenge for a specific class of RL algorithms: Policy Optimization (PO, Deisenroth et al., 2013), also known as policy search. *Policy Optimization* (PO) methods are currently the most promising for real-world applications of RL, due to their ability to deal with continuous, high-dimensional decision variables, their robustness to noise, and their general convergence properties. Compared to other RL methods, they also make it easier to exploit prior knowledge about the task at hand, which can help to ensure a minimal amount of safety to start with. Indeed, PO is behind the main achievements of RL in robotics (Kober et al., 2013; OpenAI et al., 2019; Büchler et al., 2020). Since we care about physical systems, where the safety problem is more prominent, it is only natural to focus on policy optimization algorithms.

Methodologically, policy optimization allows to apply techniques that work well for supervised learning, such as the celebrated backpropagation algorithm (stochastic gradient descent), to the RL problem. This is done by modeling the agent's behavior as a parametric mapping from states of the environment to actions (typically including an element of stochasticity for exploration purposes), called *policy*, and by defining an objective function in the policy parameters: the cumulative, expected reward collected by the agent under said parametrization. Indeed, this can be approached as a (typically nonconvex) stochastic optimization problem.

Once again, we must not be fooled into thinking this is *just* an optimization problem. The three challenges of real-world RL can be rephrased in this way from the optimization perspective:

1. Sample complexity is especially relevant given the cost of collecting interaction samples to be used by the stochastic optimization procedure. Convergence guarantees that only consider the number of parameter updates, which are a big part of optimization theory, are simply not enough.

2. Data are not provided in advance or sampled from a fixed distribution like in supervised learning, but *the very parameters that we are optimizing influence the data we are going to receive.*

3. Intermediate solutions correspond to policies that the agent uses to interact with the environment, hence have very concrete consequences. The quality of final solutions may be irrelevant by itself.

Contribution. In this dissertation, we study the problem of *safety of policy optimization algorithms*, in the online-learning, Seldonian sense outlined above, and its relation with sample complexity and exploration. We take a mostly theoretical approach, by studying the property of existing PO algorithms and seeing how the latter can be modified to provide formal guarantees. Our original contributions pertain a specific class of PO algorithms known as *Policy Gradient* (PG) methods. These are based on stochastic gradient descent, and are by far the most used by practitioners, especially due to their applicability to deep neural policies (Duan et al., 2016).[3] Our theoretical work is complemented by some algorithmic contributions, sometimes heuristic, and by empirical evaluations on simulated continuous-control problems. The latter should not be intended as the prototyping of deployable, safe reinforcement learning agents, but only as an empirical verification of the theoretical findings.

Structure and main contents. The manuscript is organized as follows. We start, in Chapter 2 by providing the fundamental concepts and methods of reinforcement learning. We do this already with a focus on continuous-action problems and solutions based on function approximation, which are the main field of application and representation tool of policy optimization, respectively. Chapter 3 is also a background chapter, but focuses on policy optimization algorithms. We provide the fundamental theoretical tools and review the most used algorithmic solutions, with a particular focus on policy gradient methods. Before diving into our technical contributions, we provide a broader view of RL safety in Chapter 4, with the purpose of better positioning our own perspective on the problem.

The following chapters present our own contributions. Chapter 5 specifically addresses our concept of safety by studying *monotonic improvement* properties of policy gradient algorithms. This is the problem of avoiding oscillations of the

[3] We only occasionally engage ourselves in *deep reinforcement learning*, since establishing theoretical guarantees for deep-learning based solutions is particularly challenging.

agent's performance during the learning process, which may correspond to undesired behavior. We follow previous work (Pirotta et al., 2013a; Papini et al., 2017) in carefully tuning the meta-parameters of the policy gradient algorithm to guarantee monotonic performance improvement. Our results are applicable to a much larger class of policies, but are more rigorous than other approaches that are not specifically taylored to policy gradient algorithms (Schulman et al., 2015a). We provide a policy gradient algorithm, called *Safe Policy Gradient* (SPG), with monotonic improvement guarantees. Note that this algorithm is mostly of theoretical interest, since it relies on worst-case assumptions which may lead to extremely conservative behavior.

In Chapter 6 we address a problem that lies in the intersection of safety and sample complexity: that of re-using historical data for policy optimization. This practice can help to reduce the need of costly interaction data, but can produce overconfident, hence unsafe behavior if performed in a naïve way. We propose an algorithm called *Policy Optimization via Importance Sampling* (POIS) which employs importance-weighted estimators to perform multiple policy updates with the same data. A thorough analysis of these off-policy estimators allows to include a penalization term in the objective of POIS which explicitly prevents overconfident learning. The fundamental intuition is to distrust data that were collected with very different policies from the one that is currently being optimized. A practical version of POIS is evaluated on benchmark continuous-control problems and is shown to be competitive with state-of-the-art PO algorithms.

Chapter 7 is straight-on about sample complexity. The problem studied here is precisely the total amount of interaction data that policy gradient algorithms require to converge to a locally optimal solution. Although not explicitly about safety, this is very relevant for any real-world application of policy optimization. Although the convergence properties of basic policy gradient algorithms such as REINFORCE (Williams, 1992) or PGT (Sutton et al., 1999) are well known, we raise awareness on some issues that may have been overlooked, such as the classes of parametric policies to which these guarantees actually apply, and the problems introduced by adaptive policy stochasticity. Then, we use insights from the stochastic optimization literature (Johnson and Zhang, 2013) to design *Stochastic Variance-Reduced Policy Gradient* (SVRPG), a policy gradient algorithm that is provably more data-efficient than REINFORCE. The superiority of SVRPG was actually proved by Xu et al. (2019), of which we report a proof. The same authors later devised an even better algorithm (Xu et al., 2020). Our presentation of SVRPG is complemented by a review of these and other related developments.

We come back to safety in Chapter 8, where we study its relationship with exploration. We focus on the specific problem of guaranteeing monotonic performance improvement when the agent also has control on the amount of stochasticity of the policy. The naïve approach would be to treat the latter as an additional policy parameter, as often done by practitioners (Duan et al., 2016). However, this would be both inefficient (since it tends to sacrifice long-term advantages in favor of immediate performance) and risky, for reasons that will be made clear throughout the dissertation. We first design a heuristic policy gradient algorithm which takes the long-term advantages of exploratory behavior explicitly into account, which may be of independent interest. Then, we provide an adaptive meta-parameter schedule

that guarantees monotonic improvement with high probability. Given the complex trade-off between safety and exploration, we leave to the user the possibility of specifying the amount of immediate performance loss they deem acceptable, or in other words, the acceptable cost of exploration.

Finally, we conclude in Chapter 9 by summarizing our contributions, mentioning some aspects of safe policy optimization that were neglected in the dissertation, discussing further possible research directions and the future of real-world RL in general.

In this Chapter, we introduce the fundamental theoretical and algorithmic tools of the RL framework, upon which all the contributions of this dissertation are built. A comprehensive introduction to the field[1] is provided by Sutton and Barto (2018).

The structure of this chapter is as follows. In Section 2.1, we describe the problem of sequential decision making under uncertainty that is the target of RL. In Section 2.2, we introduce the *Markov Decision Process* (MDP) formalism that allows to model the interaction between the agent and the environment, with a particular focus on continuous MDPs. In Section 2.3, we define policies as models of the agent's behavior and define fundamental quantities for its evaluation. In Section 2.4, we briefly present dynamic-programming solutions for known, finite MDPs. The purpose of Section 2.5 is to establish a unified notation for indefinite and finite-length interaction, since the former is more convenient for theory and the latter for experiments. In Section 2.6, we propose a possible taxonomy of RL algorithms. In Section 2.7, we present the most important policy evaluation techniques. Finally, we briefly review value-based policy learning in Section 2.8. Policy-based policy learning, or *policy optimization*, is deferred to Chapter 3.

2.1 Sequential Decison Making under Uncertainty

Reinforcement learning is an approach to the problem of taking decisions over time in an uncertain environment. The deciding entity is called an *agent*. Everything that is not under the direct control of the agent constitutes the *environment*. The agent is subject to uncertainty due to partial knowledge of the environment and/or random phenomena, such as sensor noise. This problem of *Sequential Decision*

[1] In this manuscript, we adopt the Artificial Intelligence perspective to RL. Reinforcement learning has deep roots also in the Operations Research literature, where it is historically known as *Neuro-Dynamic Programming* (Bertsekas and Tsitsiklis, 1996). See also the recent effort by Bertsekas (2019) to bridge the two research traditions.

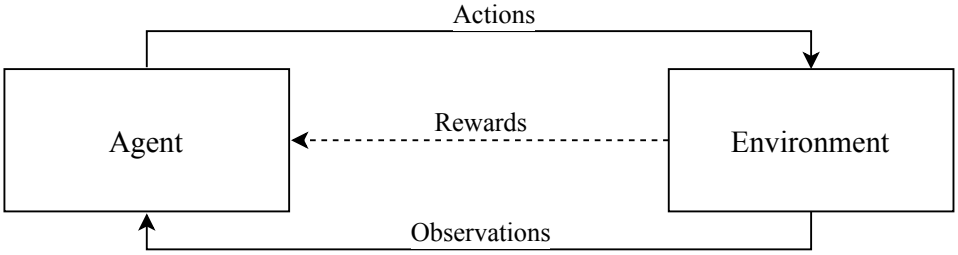


Figure 2.1: Agent-environment interaction model.

Making under Uncertainty (SDMU) is so general that solution concepts, such as RL, may be used as universal models of intelligence (Hutter, 2004). In this manuscript, we focus on artificial agents, like robots or software bots, with specific goals provided by human designers, and consider a single agent at a time. We adopt the simplest possible model of interaction between the agent and the environment, subdividing it into discrete *time steps*. These time intervals may be fixed or delimited by particular events.[2] At each step, the agent collects some information from the environment, called an *observation*, and performs an *action*. Thus, the agent faces a *decision* at each time step: which action to perform among the available ones. The decision is conditioned on the information the agent has collected so far and on the agent's *goal*, or purpose. The action that is actually performed can be subject to random errors, such as actuator noise, adding to the overall uncertainty. The agent may also incorporate a random element by design. The agent-environment interaction is illustrated in Figure 2.1.

2.1.1 The environment

We make additional assumptions on the nature of the environment and on the information accessible by the agent, which are implicit in the classical problem formalization presented in Section 2.2. By *state of the environment* we mean a complete description of the environment at a given time. We borrow most of the following nomenclature from Russell and Norvig (2010):

- ◇ *Stationarity*: the state of the environment only changes as a result of the agent's actions.
- ◇ *Effectively full observability*: the agent can access all the information about the state of the environment that is relevant to decision-making.
- ◇ *Stochasticity*: the change in the state of the environment can be modeled with a probability distribution.

[2] In this manuscript, we take this design choice for granted and work under the time-step abstraction. Generalization of this simplistic model include continuous-time RL, temporal abstractions and, more recently, action persistence (Metelli et al., 2020a).

- ◇ *Unknowing*: the agent does not know in advance the laws regulating the change in the state of the environment.

The first three are simplifying assumptions. Nonstationary, partially observable and adversarial environments are active areas of RL research (Choi et al., 1999; Jaakkola et al., 1994b; Busoniu et al., 2008). However, we think many interesting real-world problems can already be modeled under these assumptions. The fourth assumption is what makes *learning* necessary. Under knowledge of the environment laws, SDMU reduces to a (possibly still very difficult) planning problem.

2.1.2 Learning agents

From the engineering perspective, the goal of artificial intelligence is to build rational artificial agents. A *rational* agent always selects the best possible action with respect to its goal, given the currently available knowledge. In an unknown environment, this requires the agent to improve as more information is collected, and to actively pursue the acquisition of useful information. A *learning* agent is one that improves from experience, always with respect to its goal.

We assume that the agent is able to construct, from observations, an *agent state* that encodes all the information necessary for rational decision making. This is only possible in effectively fully observable environments. The agent state will be called simply *state* from now on, and must not be confused with the environment state, of which it is a byproduct. We require the state to satisfy *Markov's property*: the next state, conditioned on the current state and the agent's action, is independent from all previous states and actions. That is, the state summarizes all the information on past experience that is relevant to prediction and decision-making. State definition and synthesis is part of agent design, but is not addressed in this work. Similarly, we take the definition of the actions available to the agent as granted.[3]

2.1.3 Exploration vs exploitation

A key challenge of SDMU is to find the right balance between *exploration*, acting as to collect more information about the unknown environment, and *exploitation*, acting rationally with respect to the information collected so far. This is known as the *exploration-exploitation dilemma*. Exploration typically requires to diversify the agent's actions, exploitation to focus on what appear to be the best ones. In a safety-critical scenario, exploration also contrasts with safety, especially if the agent resorts to action randomization to guarantee a sufficient amount of exploration. This *safe exploration* problem is introduced in Chapter 4 and developed in Chapter 8.

2.1.4 Goals as rewards

The distinctive assumption underlying the RL approach to SDMU is that an agent's goal can always be encoded in a scalar signal, called *reward*. This measure of goal-

[3] In doing so, we are certainly neglecting important engineering problems. A recent work on robot table tennis by Büchler et al. (2020) showed how an unconventional choice of actuators can already remove some safety issues.

adherence is to be provided to the agent at each time step depending on the current state of the environment and on the agent's action. Negative rewards correspond, intuitively, to punishments. No matter how complex the goal, there exists a reward signal such that maximizing the sum of rewards is equivalent to achieving the goal. This conjecture is known as the *Reward Hypothesis* (Sutton and Barto, 2018). Deriving a proper reward signal from an abstract goal or from a reference behavior can be challenging, especially in the presence of multiple, contrastive objectives, which are typical of safety-critical scenarios. Modern approaches to the problem include Inverse Reinforcement Learning (Ng and Russell, 2000). In this work, we do not actively address the reward-design problem. However, in Section 4.2, we briefly discuss the potentially dangerous effects of reward misspecification. Most importantly, the Monotonic-Improvement approach to safe RL, introduced in Chapter 1 and presented in detail in Chapter 4, by assuming that safety concerns are encoded in the reward signal, heavily relies on the Reward Hypothesis.

2.2 Markov Decision Processes

The interaction between the agent and the environment and the SDMU problem faced by the agent, as outlined in the previous section, can be formalized as a *Markov Decision Process* (MDP). The reference book on MDPs is (Puterman, 2005).

A MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma, \mu \rangle$ where:

- $\mathcal{S}$ is the *state space*, a measurable set containing all possible (agent) states.
- $\mathcal{A}$ is the *action space*, a measurable set containing all the actions available to the agent in any state.[4]
- $p : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ is the *transition kernel*, providing a probability measure over the state space for each state-action pair. Given $s \in \mathcal{S}$ and $a \in \mathcal{A}$, $p(\cdot \mid s, a)$ denotes the probability measure of the *next state*. Overloading the notation, $p(s' \mid s, a)$ denotes the probability (in the sense of a density or mass function) of transitioning to $s' \in \mathcal{S}$ when performing action a in state s.
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *reward function*. We always assume it is bounded, i.e., $\|r\|_\infty < \infty$.
- $\gamma \in [0, 1)$ is the *discount factor* used to weight future rewards.
- $\mu \in \Delta_{\mathcal{S}}$ is the *starting-state distribution*, from which the initial state is sampled.

2.2.1 Continuous state and action spaces

The literature on MDPs has mainly focused on finite state and action spaces. However, many interesting control problems, such as robotics ones, are naturally modeled with continuous states and actions, making $\mathcal{S}$ and $\mathcal{A}$ subsets of $\mathbb{R}$ (or

[4] The set of available actions may be a function of the state. We assume a single, common action space for simplicity.

of higher-dimensional real coordinate spaces for vector-valued states and actions). This is the natural setting of *policy optimization* algorithms, which are the focus of this manuscript. For this reason, we generally assume continuous states and actions are involved, write expectations and marginalizations as integrals,[5] and overload the symbol for a measure (e.g., p or μ) to denote a probability density function when it is applied to a point. In the less frequent case of finite state and action spaces, one can safely replace integrals with summations and interpret the overloaded measure symbol as a probability mass function. Continuous measurable spaces come with measurability and integrability issues, starting from the very definition of MDP (Puterman, 2005). We have neglected those for ease of exposition.

2.2.2 The reinforcement learning objective

Fixed a MDP, the goal of the agent is to maximize the sum of rewards, in expectation over all the sources of randomness. In most of our theoretical contributions, we consider the following model of interaction: the initial state is sampled from the starting state distribution, $s_0 \sim \mu$; then, at each time step t, the agent performs an action a_t, receives a reward $r_{t+1} = r(s_t, a_t)$ and the next state is sampled from the transition kernel, $s_{t+1} \sim p(\cdot \mid s_t, a_t)$. We consider an infinite time horizon and adopt the *discounted-reward formulation*.[6] The RL objective is then:

$$\max_{a_0, a_1, \dots} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s_0 \sim \mu, s_{t+1} \sim p(\cdot \mid s_t, a_t) \text{ for all } t > 0 \right]. \tag{2.1}$$

This is a *multi-stage* stochastic optimization problem, since the effects of each decision a_t can arbitrarily unravel in time, influencing all future rewards through the changing state. Besides making the infinite sum of rewards well defined, the discount factor γ can be interpreted either as assigning a lower value to rewards further in the future, smaller values of γ corresponding to a more short-sighted objective. Although it is often used as a hyper-parameter in RL algorithms, the discounting γ^t is in fact part of the reward *signal*, and different values of γ encode different goals, resulting in more or less far-sighted target behavior.

2.3 Policies

To solve (2.1), the agent need not explicitly optimize over actions plans $(a_0, a_1, \dots)$, but only find an optimal *stationary deterministic policy*, that is a mapping from states to actions. This is a consequence of Markov's property and is stated more formally in Section 2.3.1. A *stationary deterministic policy* is any function $\pi : \mathcal{S} \to \mathcal{A}$, and $\pi(s)$ denotes the action prescribed by policy π in state $s \in \mathcal{S}$. A *learning* agent keeps updating its policy based on experience. Since RL agents often incorporate a random element for exploration, we must also consider *stationary stochastic policies*

[5] These must be intended as Lebesgue integrals where the Lebesgue measure (or the counting measure) is implicit and all the other involved measures are absolutely continuous w.r.t. it.

[6] Other possible forumulations are *average-reward* and *finite-horizon*.

of the form $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$, providing a probability measure $\pi(\cdot|s)$ over the action space for each state $s \in \mathcal{S}$. When π is stochastic, $\pi(a|s)$ denotes the probability density of action a in state s.

2.3.1 Value functions and optimal policies

Value functions allow to translate the multi-stage optimization problem in (2.1) into a fixed-point problem by exploiting Markov's property. The results presented here are standard in the RL literature. Refer to Bertsekas and Shreve (2004) for a more complete treatment and proofs.[7] Fix an MDP and let $T^\pi : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$ be the *Bellman Expectation Operator* of policy π, defined as follows:

$$T^\pi(V)(s) = \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}\left[r(s,a) + \gamma \mathop{\mathbb{E}}_{s'\sim p(\cdot|s,a)}\left[V(s')\right]\right], \tag{2.2}$$

for all $V : \mathcal{S} \to \mathbb{R}$ and $s \in \mathcal{S}$. The *value function* $V^\pi : \mathcal{S} \to \mathbb{R}$ of policy π is the unique fixed point of T^π and satisfies *Bellman's expectation equation*:

$$V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}\left[r(s,a) + \gamma \mathop{\mathbb{E}}_{s'\sim p(\cdot|s,a)}\left[V^\pi(s')\right]\right], \tag{2.3}$$

for all $s \in \mathcal{S}$. Equivalently, the value of a state $V^\pi(s)$ is the expected, discounted sum of rewards obtained by following policy π starting from s:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\,\middle|\, s_0 = s, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t,a_t) \text{ for all } t > 0\right] \\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)|s_0 = s\right], \end{aligned}$$

where $\mathbb{E}_\pi[\cdot]$ is an abbreviation for *expectation following policy* π that we will often use when it does not introduce any ambiguity.

The *quality function*, or action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined as follows:

$$Q^\pi(s,a) = r(s,a) + \gamma \mathop{\mathbb{E}}_{s'\sim p(\cdot|s,a)}\left[V^\pi(s')\right], \tag{2.4}$$

for all $s \in \mathcal{S}, a \in \mathcal{A}$, and assigns to each state-action pair, i.e., to each decision, the expected, discounted sum of rewards obtained by performing action a in state s and following policy π thereafter. Note that $V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^\pi(s,a)\right]$.

The *advantage function* $A^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ of policy π is simply the difference between the quality function and the value function:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s), \tag{2.5}$$

[7] More common references on the topic are (Bertsekas and Tsitsiklis, 1996) and (Puterman, 2005), but they are mostly focused on finite-space MDPs. Note that all the fundamental results on finite MDPs transfer to the continuous domain once measurability issues are properly dealt with.

for all $s \in \mathcal{S}, a \in \mathcal{A}$.

Let $T^* : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$ be *Bellman's Optimality Operator*, defined as follows:

$$T^*(V)(s) = \max_{a \in \mathcal{A}} \left\{ r(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim p(\cdot|s,a)} \left[V(s') \right] \right\}, \tag{2.6}$$

for all $V : \mathcal{S} \to \mathbb{R}$ and $s \in \mathcal{S}$. The *optimal value function* $V^* : \mathcal{S} \to \mathbb{R}$ is the unique fixed point of T^* and satisfies *Bellman's optimality equation*:

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ r(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim p(\cdot|s,a)} \left[V^\pi(s') \right] \right\}, \tag{2.7}$$

for all states $s \in \mathcal{S}$. We can easily define the *optimal quality function* as $Q^*(s,a) = r(s,a) + \gamma \, \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[V^*(s') \right]$. Now consider the following deterministic policy, greedy w.r.t. the optimal quality function:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a), \tag{2.8}$$

with ties broken in any (deterministic) way. This is an optimal policy for the MDP and has the following properties:

- $V^{\pi^*} = V^*$, i.e., the optimal value function is the value function of the optimal policy. Also $Q^{\pi^*} = Q^*$.
- $V^* \geq V^{\pi'}$ for all π', i.e., the optimal policy maximizes the value function simultaneously in all states.

For any MDP, there always exists at least one deterministic (stationary) optimal policy. It may not be unique, and there may also exist optimal stochastic policies.

2.3.2 Occupancy measures

Fixed a policy π, we can define the following induced transition kernel by marginalizing over actions:

$$p_\pi(\cdot|s) \coloneqq \int_{\mathcal{A}} \pi(a|s) p(\cdot|s,a) \, \mathrm{d}a. \tag{2.9}$$

We can define a t-step transition kernel recursively as follows:

$$p_\pi^1(\cdot|s) \coloneqq p_\pi(\cdot|s), \tag{2.10}$$

$$p_\pi^{t+1}(\cdot|s) \coloneqq \int_{\mathcal{S}} p_\pi^t(s'|s) p_\pi(\cdot|s') \, \mathrm{d}s', \tag{2.11}$$

for all $s \in \mathcal{S}$ and $t \geqslant 1$. From this, we can define, for any state s_0, the following γ-discounted state-occupancy measure:

$$d_{s_0}^\pi(\cdot) = \frac{1-\gamma}{\gamma} \sum_{t=1}^{\infty} \gamma^t p_\pi^t(\cdot|s_0). \tag{2.12}$$

Intuitively, $d^\pi_{s_0}(s)$ is the discounted probability of visiting state s *at some point in the future*, starting from s_0 and following π. The following fact about $d^\pi_{s_0}$, which is a variant of the *generalized eigenfunction property* by Ciosek and Whiteson (2020, Lemma 20), will be used multiple times:

Lemma 2.1 *Let $\pi \in \Delta_{\mathcal{A}}^{\mathcal{S}}$ and f be any integrable function on $\mathcal{S}$ satisfying the following recursive equation:*

$$f(s) = g(s) + \gamma \int_{\mathcal{S}} p_\pi(s'|s) f(s') \,\mathrm{d}s',$$

for all $s \in \mathcal{S}$ and some integrable function g on $\mathcal{S}$. Then:

$$f(s) = g(s) + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} d^\pi_s(s') g(s') \,\mathrm{d}s',$$

for all $s \in \mathcal{S}$.

Proof

$$\begin{aligned}
\int_{\mathcal{S}} d^\pi_s(s') g(s') \,\mathrm{d}s' &= \int_{\mathcal{S}} d^\pi_s(s') f(s') \,\mathrm{d}s' - \int_{\mathcal{S}} d^\pi_s(s') \gamma \int_{\mathcal{S}} p_\pi(s''|s') f(s'') \,\mathrm{d}s'' \,\mathrm{d}s' \\
&= \int_{\mathcal{S}} d^\pi_s(s') f(s') \,\mathrm{d}s' - \int_{\mathcal{S}} \gamma \int_{\mathcal{S}} d^\pi_s(s') p_\pi(s''|s') \,\mathrm{d}s' f(s'') \,\mathrm{d}s'' \\
&= \int_{\mathcal{S}} d^\pi_s(s') f(s') \,\mathrm{d}s' - \int_{\mathcal{S}} \left(d^\pi_s(s'') - (1-\gamma) p_\pi(s''|s) \right) f(s'') \,\mathrm{d}s'' \qquad (2.13) \\
&= (1-\gamma) \int_{\mathcal{S}} p_\pi(s''|s) f(s'') \,\mathrm{d}s'',
\end{aligned}$$

where (2.13) is from Lemma B.2. ■

For instance, V^π satisfies the assumptions of Lemma 2.1 with $g(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a)]$, allowing to write the value function as follows:

$$V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}[r(s,a)] + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} d^\pi_s(s') \mathop{\mathbb{E}}_{a'\sim\pi(\cdot|s')} \left[r(s',a') \right] \mathrm{d}s'. \qquad (2.14)$$

A further γ-discounted state-occupancy measure is obtained by weighting the starting state with the starting-state distribution μ of the MDP:

$$d^\pi_\mu(\cdot) \coloneqq (1-\gamma)\mu(\cdot) + \gamma \int_{\mathcal{S}} \mu(s_0) d^\pi_{s_0}(\cdot) \,\mathrm{d}s_0. \qquad (2.15)$$

Intuitively, $d^\pi_\mu(s)$ is the discounted probability of visiting state s *at some point of the interaction* (possibly at the start) by following policy π. In the following, we will call d^π_μ simply the *state-occupancy distribution* of π.

By composing d^π_μ with the policy itself, we obtain a measure of the frequency of state-action pairs:

$$\nu^\pi_\mu(s,a) \coloneqq d^\pi_\mu(s)\pi(a|s). \qquad (2.16)$$

We call this the *state-action-occupancy distribution* of π.

2.4 Dynamic Programming

Given a *finite* MDP, under perfect knowledge of the transition kernel and of the reward function, one can use *dynamic programming* to find an optimal policy (Bertsekas and Tsitsiklis, 1996).

2.4.1 Policy evaluation

The *policy evaluation* problem is important both by itself (as a *prediction* task) and as a building block for finding optimal policies. Given a policy π, the problem is just to compute its value function V^π. The solution of *iterative policy evaluation* is to repeatedly apply Bellman's Expectation Operator T^π (2.2) to a tentative value function $V : \mathcal{S} \to \mathbb{R}$. Starting from an arbitrary value function V_0 (e.g., a vector of zeros), the next estimate is computed as:

$$V_{k+1} = T^\pi V_k, \tag{2.17}$$

for $k = 0, 1, \ldots$ until convergence. See Sutton and Barto (2018, Section 4.2) for a discussion of this algorithm. However, by rewriting Bellman's expectation equation as a system of linear equations, we see that several other solutions are possible. Represent value functions as vectors in $\mathbb{R}^{\mathcal{S}}$. Let $P^\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ be the matrix form of the induced transition kernel p_π (2.9) and $\mathbf{r}^\pi \in \mathbb{R}^{|\mathcal{S}|}$ be the vector form of $s \mapsto \mathbb{E}_{a\sim\pi(\cdot|s)}[r(s,a)]$. Bellman's expectation equation then rewrites as the following system of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns:

$$(I - \gamma P^\pi)V^\pi = \mathbf{r}^\pi, \tag{2.18}$$

which admits a unique solution (Puterman, 2005, Theorem 6.1.1). Hence, any method for solving linear systems can be used to compute V^π. The iterative policy evaluation algorithm outlined above corresponds to the Jacobi method,[8] as observed by Sutton and Barto (2018). From V^π, we can easily obtain Q^π by using its definition (2.4).

2.4.2 Policy iteration

Going from prediction to *control*, dynamic programming provides ways to compute an optimal policy. *Policy iteration* starts from an arbitrary policy π_0 and alternates between the two steps below:

1. Policy evaluation: given π_k, compute Q^{π_k};
2. Policy improvement: $\pi_{k+1}(\cdot) \leftarrow \arg\max_a Q^{\pi_k}(\cdot, a)$,

for $k = 0, 1, \ldots$ until convergence. The policy evaluation step may itself be an iterative procedure (see Section 2.4.1). The *policy improvement* step simply returns a greedy policy w.r.t. Q^{π_k}, which can be shown to be an improvement over π_k itself (Puterman, 2005, Proposition 6.4.1). Policy iteration yields an optimal policy in a finite number of iterations (Puterman, 2005, Theorem 6.4.2).

[8] The system matrix $I - \gamma P^\pi$ is strictly diagonally dominant.

2.4.3 Value iteration

A more direct way to compute the optimal policy is to iterate the application of Bellman's Optimality Operator T^* (2.6). Starting from an arbitrary value function V_0, the next iterate is computed as:

$$V_{k+1} = T^* V_k, \tag{2.19}$$

for $k = 0, 1, \ldots$ until convergence. The iterates converge *in infinity norm* to the optimal value function V^*. In practice, one can fix a small threshold $\epsilon > 0$ and stop as soon as $\|V_k - V^*\|_\infty < \epsilon$, which is guaranteed to happen within a finite number of steps K. From V_K, one can compute the corresponding quality function, using (2.4), and the greedy policy w.r.t. Q_k, denoted π_K. This is guaranteed to be an ϵ-*optimal policy*, that is:

$$V^{\pi_K} \geqslant V^* - \epsilon. \tag{2.20}$$

See Puterman (2005, Theorem 6.3.1) for a complete proof.

Dynamic programming cannot be employed if the transition kernel is unknown, which is always the case for the problems of our interest. Moreover, no trivial extension to continuous-space MDPs (or even MDPs with very large state spaces) is available. Still, the fundamental ideas of dynamic programming play a major role in many RL methods. Furthermore, several extensions and optimizations of the dynamic programming algorithms presented here are known in the operations research literature under the name of *Approximate Dynamic Programming*, with significant overlapping with Reinforcement Learning (Bertsekas, 2019). That said, we will present RL algorithms from a Machine Learning perspective starting from Section 2.6.

2.5 Interaction in Practice

The model of interaction depicted in Section 2.2 is that of a single, infinite stream. In practice, the training of RL agents is more often performed in a series of finite-length episodes. Each episode may have a predetermined length or its end may be triggered by the occurence of some event (e.g., the fall of a self-balancing robot). A new episode is then started with the initial state sampled from μ. This starting-state distribution may be partially or entirely under the control of a human operator (e.g., someone in charge of repositioning the fallen robot). This kind of interaction is known as *episodic*, as opposed to the *continuing* (never-ending) interaction depicted in Section 2.2. Moreover it is often argued that, in practice, one cares about maximizing the *undiscounted* reward. This would correspond to a discount factor of $\gamma = 1$, which is not compatible with the formulation we have chosen to adopt.

2.5.1 Indefinite trajectories

We can accomodate both episodic interaction and undiscounted rewards within the discounted formulation by interpreting the discount factor as a continuation probability. Under this perspective, a discount factor of γ means that, at each time step t, the current episode terminates with probability $1-\gamma$ (just after observing reward r_{t+1}), in which case a new starting state is sampled from μ. Hence, we can consider two alternative models of interaction under a unified theoretical formulation:

1. Continuing interaction with rewards discounted by $\gamma \in [0, 1)$;
2. Episodic interaction with undiscounted rewards and continuation probability $\gamma \in [0, 1)$.

The key feature of this double interpretation is that the state-occupancy measure d_μ^π defined in (2.15) can be interpreted either as the γ-discounted distribution of states under a single continuing interaction or as the distribution of states in episodic interaction with continuation probability $\gamma < 1$. Also the value $V^\pi(s_t)$, as defined in (2.3), has a special interpretation in the episodic view: it is the expected sum of rewards from s_t to the end of the current episode.

A *trajectory* is a sequence $\tau = (s_0, a_0, r_1, s_1, a_1, r_2 \dots)$ where $s_0 \sim \mu$, $a_t \sim \pi(\cdot|s_t)$, $r_{t+1} = r(s_t, a_t)$, and $s_{t+1} \sim p(\cdot|s_t, a_t)$ for all $t \geqslant 0$, for some policy π. In continuing interaction, this is one of the possible infinite sequences that can be generated by policy π. In episodic interaction, trajectories have finite length almost surely. To see that trajectories are still i.i.d. in the episodic case, consider the following *equivalent* data-generating process:

1. Sample $H \sim \text{Geom}(1-\gamma)$;
2. Run an episode of length H,

from which it is apparent that we have a probability of $\gamma^H(1-\gamma)$ of generating a trajectory of length H. These are *indefinite* trajectories as their length is not predetermined. We can define the distribution induced by policy π over indefinite trajectories as:

$$p_{\pi,\gamma}(\tau) = \gamma^{|\tau|-1}(1-\gamma)\mu(s_0)\left(\prod_{t=0}^{|\tau|-2}\pi(a_t|s_t)p(s_{t+1}|s_t,a_t)\right)\pi\left(a_{|\tau|-1}|s_{|\tau|-1}\right), \quad (2.21)$$

where $|\tau|$ denotes the length (number of states) of trajectory τ.

We use the notation $\tau_{h:k}$, with $0 \leqslant h \leqslant k < |\tau|$, to denote a partial trajectory $(s_h, a_h, s_{h+1} \dots, s_k, a_k)$, where the elements are from trajectory τ. In particular, we use the following notation for taking expectations:

$$\mathop{\mathbb{E}}_{\tau_{0:k}\sim p_\pi}[X] = \int_{\mathcal{S}} \mu(s_0) \dots \int_{\mathcal{A}} \pi(a_k|s_k) X \, \mathrm{d}a_k \dots \, \mathrm{d}s_0, \quad (2.22)$$

$$\mathop{\mathbb{E}}_{\tau_{h:k}\sim p_\pi}[X] = \int_{\mathcal{S}} p(s_h|s_{h-1}, a_{h-1}) \dots \int_{\mathcal{A}} \pi(a_k|s_k) X \, \mathrm{d}a_k \dots \, \mathrm{d}s_h \quad \text{if } h > 0, \quad (2.23)$$

where the second expectation is implicitly conditioned on s_{h-1} and a_{h-1}.

The following lemma clarifies the relationship between trajectory distributions and the occupancy measure:

Lemma 2.2 *For any policy π and integrable function f on $\mathcal{S}$:*

$$\mathop{\mathbb{E}}_{\tau\sim p_{\pi,\gamma}}\left[\sum_{t=0}^{|\tau|-1} f(s_t)\right] = \sum_{t=0}^{\infty}\gamma^t \mathop{\mathbb{E}}_{\tau_{0:t}\sim p_\pi}\left[f(s_t)\right] = \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{s\sim d_\mu^\pi}\left[f(s)\right].$$

Proof For the first equality:

$$\begin{aligned}
\mathop{\mathbb{E}}_{\tau\sim p_{\pi,\gamma}}\left[g(\tau)\right] &= (1-\gamma)\sum_{H=1}^{\infty}\gamma^{H-1}\mathop{\mathbb{E}}_{\tau_{0:H-1}\sim p_\pi}\left[\sum_{t=0}^{H-1} f(s_t)\right] \\
&= (1-\gamma)\sum_{H=1}^{\infty}\gamma^{H-1}\sum_{t=0}^{H-1}\mathop{\mathbb{E}}_{\tau_{0:t}\sim p_\pi}\left[f(s_t)\right] \\
&= \sum_{t=0}^{\infty}\gamma^t \mathop{\mathbb{E}}_{\tau_{0:t}\sim p_\pi}\left[f(s_t)\right], \qquad (2.24)
\end{aligned}$$

where (2.24) is from the following equality that holds for any bounded $(a_j)_{j=0}^{\infty}$:

$$\begin{aligned}
\sum_{i=0}^{\infty}\gamma^i\sum_{j=0}^{i}a_j &= \sum_{j=0}^{\infty}\left(\sum_{i=j}^{\infty}\gamma^i\right)a_j \qquad (2.25)\\
&= \sum_{j=0}^{\infty}\gamma^j\left(\sum_{i=0}^{\infty}\gamma^i\right)a_j \\
&= \frac{1}{1-\gamma}\sum_{j=0}^{\infty}a_j, \qquad (2.26)
\end{aligned}$$

and (2.25) is from reordering (see Lemma B.1). For the second equality:

$$\begin{aligned}
&\sum_{t=0}^{\infty}\gamma^t\mathop{\mathbb{E}}_{\tau_{0:t}\sim p_\pi}\left[f(s_t)\right] = \int_{\mathcal{S}}\left[\mu(s)+\int_{\mathcal{S}}\mu(s_0)\sum_{t=1}^{\infty}\gamma^t p_\pi^t(s|s_0)\,\mathrm{d}s_0\right]f(s)\,\mathrm{d}s \\
&= \frac{1}{1-\gamma}\int_{\mathcal{S}}\left[(1-\gamma)\mu(s)+\gamma\int_{\mathcal{S}}\mu(s_0)d_{s_0}^\pi(s)\,\mathrm{d}s_0\right]f(s)\,\mathrm{d}s \\
&= \frac{1}{1-\gamma}\int_{\mathcal{S}} d_\mu^\pi(s)f(s)\,\mathrm{d}s, \qquad (2.27)
\end{aligned}$$

by definition of $d_{s_0}^\pi$ (2.12) and d_μ^π (2.15). ■

2.5.2 Finite trajectories

The present formulation does not yet account for the case, very common in practice, in which episodes have a fixed maximum length H, called *time horizon*. To model this situation, we introduce the concept of *terminal state*:

Definition 2.5.1 *A terminal state $s \in \mathcal{S}$ is such that $r(s,a) = 0$ and $p(s'|s,a) = 0$ for all $a \in \mathcal{A}$ and $s \neq s'$,*

that is just a self-looping state with zero rewards. Note that $V^\pi(s) = 0$ for all π if s is a terminal state. When we want to consider a maximum horizon H, we make the following assumption:

Assumption 2.1 *For all $\pi \in \Delta_{\mathcal{A}}^{\mathcal{S}}$, from any state in $\mathcal{S}$, the agent reaches a terminal state within H steps.*

This framework has three advantages: we can completely neglect what happens after H steps from the beginning of the episode (both for prediction and control purposes); we can normalize trajectories to have a fixed length H (completing with zeros and ones where necessary); we can still interpret γ either as a continuation probability (only relevant before H) or as a discount factor, and the value function is well defined even if we set $\gamma = 1$.

The *return to go* from state s_t (obtained following policy π):

$$G_t = \sum_{h=0}^{H-1} \gamma^h r_{t+h}, \tag{2.28}$$

is an unbiased estimate of the value, i.e., $V^\pi(s_t) = \mathbb{E}_\pi[G_t]$. The *return* of a trajectory is simply:

$$R(\tau) = G_0 = \sum_{t=0}^{H-1} \gamma^t r_{t+1}. \tag{2.29}$$

We can also define the distribution over finite trajectories (normalized to have length H) induced by policy π as:

$$p_{\pi,H}(\tau) = \mu(s_0) \left(\prod_{t=0}^{H-2} \pi(a_t|s_t) p(s_{t+1}|s_t,a_t) \right) \pi\left(a_{H-1}|s_{H-1}\right). \tag{2.30}$$

Note that $\mathbb{E}_{\tau \sim p_{\pi,H}}[\cdot]$ is equivalent to $\mathbb{E}_{\tau_{0:H-1} \sim p_\pi}[\cdot]$ in our notation. We will just denote the distribution over trajectories as p_π when this does not introduce any ambiguity.

When Assumption 2.1 is not actually satisfied (e.g., if termination after H steps is enforced by a human operator regardless of the agent's state), we accept some degree of approximation.[9] To quantify this imprecision, we introduce the concept of *effective horizon*:

[9] Explicit finite-time RL would require its own theoretical framework (cf. Puterman, 2005, Chapter 4). For instance, stationary policies are no longer optimal in this setting.

Definition 2.5.2 *The effective horizon associated to episodic interaction with continuation probability γ is $H_\gamma = 1/(1-\gamma)$.*

As shown by Kearns and Singh (2002, Lemma 2):

Lemma 2.3 *Fix an $\epsilon > 0$. Assume that the reward function is uniformly bounded as $\|r\|_\infty \leqslant R_{\max}$ and $H \geqslant H_\gamma \log(R_{\max}/(\epsilon(1-\gamma)))$. Then, for all policies $\pi \in \Delta_{\mathcal{A}}^{\mathcal{S}}$ and states $s \in \mathcal{S}$:*

$$0 \leqslant V^\pi(s) - \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{H-1} \gamma^t r_{t+1} \mid s_0 = s\right] \leqslant \epsilon. \tag{2.31}$$

In practice, we just make sure to set the discount factor to $\gamma \simeq 1 - 1/H$. Lemma 2.3, together with $H \simeq H_\gamma$, ensures that we are not introducing large errors.

2.6 A Taxonomy of Reinforcement Learning Algorithms

In this section, we try to organize existing RL algorithms according to several criteria, some pertaining the properties of the algorithms themselves, some the assumptions under which they operate. The chosen categories are largely orthogonal and do not have the presumption to be exhaustive. In fact, not all RL algorithms can be clearly classified along all the proposed dimensions.

The first distinction is about the objective of the algorithm, and mirrors one already made for dynamic programming in Section 2.4:

- ◇ *Evaluation* (or prediction) algorithms aim to estimate the performance of a given policy;
- ◇ *Optimization* (or control) algorithms aim to find an optimal policy, or at least to improve a given one.

This manuscript is about optimization algorithms, but policy evaluation is an important building block.

The fundamental difference between dynamic programming and RL is that in the latter we do not have direct access to the transition kernel. The next distinction is about the kind of indirect access that is available:

- ◇ *Generative model*: we can sample from the transition kernel $p(\cdot|s,a)$ (typically a software simulator) the next state for any given $s \in \mathcal{S}$ and $a \in \mathcal{A}$.
- ◇ *Online interaction*: we can only interact with the environment in a sequential manner, as outlined in Section 2.2.2. Alternative models of interaction, for instance in finite trajectories, are discussed in Section 2.5.
- ◇ *Batch RL*: we have only access to a log of state-action-next-state (and reward) tuples, for instance collected in past interaction. The High Confidence Policy Improvement algorithm by Thomas et al. (2015b), mentioned in Section 4.5.1, is an example of batch RL algorithm.

Online interaction is the most realistic assumption for the problems of our interest. A similar distinction can be done for the reward. Most of the algorithms we present assume *bandit feedback* (we only observe the rewards of chosen actions), but in many applications (especially in robotics) the reward function is designed beforehand by a human, hence is perfectly known.

This indirect source of information is what makes learning necessary. A third distinction is on the kind of object that is learned:

- ◇ *Model-based RL*: we learn a model of the environment (the transition kernel and possibly the reward function), which can then be used for optimization purposes;
- ◇ *Value-based RL*: we learn value functions, from which policies (e.g., greedy ones) can then be computed;
- ◇ *Policy-based RL*: we learn policies directly.

Model-based algorithms are relevant both w.r.t. data efficiency and safety, but lie beyond the scope of this manuscript. See Nguyen-Tuong and Peters (2011) for a survey. This manuscript is about *model-free*[10] policy-based optimization algorithms, and Chapter 3 is entirely dedicated to them. Value-based and policy-based algorithms are often compared and sometimes strongly related, so we briefly review model-free value-based RL in Section 2.8.

Both in value-based and policy-based control, we need to specify the relationship between the data-generating process and the learning target:

- ◇ *On-policy RL*: the policy that is (or has been) used to collect data is the same that is (directly or indirectly) being learned;
- ◇ *Off-policy RL*: data are generated by one or more *behavioral policies* that differ from the *target policy* that is the object of learning.

Both approaches apply to policy optimization and will be thoroughly examined in the following chapters.

Finally, we distinguish the kind of representation that is adopted for learning:

- ◇ *Tabular RL*: information is stored in tables (finite vectors, matrices, tensors). This is only possible in finite MDPs, or after discretization;
- ◇ *Function approximation*: policy and/or values are represented by parametric functions, and we only learn the parameters. These methods can be applied to continuous-space MDPs directly. The term *Deep Reinforcement Learning* refers to the special case in which multi-layer neural networks are used as approximators. *Non-parametric* approaches are also possible.

Given our focus on continuous MDPs, function approximation will play a major role in our treatment of policy optimization algorithms.

[10] This simply means that we do *not* learn an *explicit* model of the environment.

The starting point of much of our original contributions, REINFORCE (Williams, 1992), is a policy-based control algorithm based on (episodic) online interaction, where the policy is typically represented with function approximation.

2.7 Reinforcement Learning for Policy Evaluation

In this section we provide a brief review of model-free policy evaluation algorithms. This is not meant as a comprehensive coverage of the topic (see Sutton and Barto (2018) for a good starting point), but is entirely ancillary to the later discussion of policy optimization algorithms, which often employ policy evaluation as a building block.

We will focus on the problem of approximating the value function V^π of a given policy π in an unknown, continuous-space MDP. This must not be confused with the easier problem of *value estimation*, where we only need to estimate the value of a given state, $V^\pi(s)$. In policy evaluation, we want to (approximately) reconstruct the whole value function. Let $V^{\boldsymbol{\omega}} : \mathcal{S} \to \mathbb{R}$ be an approximation of V^π, parametrized by a vector $\boldsymbol{\omega} \in \Omega \subseteq \mathbb{R}^d$ of dimension d. Assume that $V^{\boldsymbol{\omega}}$ is differentiable w.r.t. $\boldsymbol{\omega}$. The policy evaluation problem can be formulated as the minimization of the following *Mean Squared Error* (MSE):

$$L(\boldsymbol{\omega}) = \frac{1}{2} \mathop{\mathbb{E}}_{s \sim d_\mu^\pi} \left[\left(V^\pi(s) - V^{\boldsymbol{\omega}}(s) \right)^2 \right]. \tag{2.32}$$

Function approximation can introduce an irreducible bias if the true value function does not belong to the function class $\mathcal{V}_\Omega = \{V^{\boldsymbol{\omega}} : \boldsymbol{\omega} \in \Omega\}$:

$$\text{Bias}^2(\mathcal{V}_\Omega) = \min_{\boldsymbol{\omega} \in \Omega} L(\boldsymbol{\omega}). \tag{2.33}$$

If $V^\pi \in \mathcal{V}_\Omega$ the bias is zero and we say that the approximation is *realizable*.

For finite MDPs, the tabular representation is recovered by representing the value function estimate as a vector parametrized by the elements themselves and indexed by states.[11] Minimizing $L(\boldsymbol{\omega})$ is equivalent to minimizing the error $|V^\pi(s) - V^{\boldsymbol{\omega}}(s)|$ uniformly over $\mathcal{S}$ in the tabular case, provided the Markov chain induced by π is *irreducible*.

In continuous-space MDPs, a generalization of the tabular representation is *linear* function approximation:

$$V^{\boldsymbol{\omega}}(s) = \boldsymbol{\omega}^T \phi(s), \tag{2.34}$$

for some feature function $\phi : \mathcal{S} \to \mathbb{R}^d$. Linear approximation is exact only for *linear MDPs* with matching features (Cai et al., 2019a).

[11] The ∇ operator can be interpreted as an element selector.

2.7.1 Monte Carlo evaluation

In the case of episodic interaction (assume time horizon H), the return-to-go G_t is an unbiased estimate of $V^\pi(s_t)$. So, we can use G_t as a target for V^π:

$$\boldsymbol{\omega}_{t+1} \leftarrow \boldsymbol{\omega}_t + \underbrace{\alpha_t \left(G_t - V^{\boldsymbol{\omega}_t}(s_t)\right) \nabla_{\boldsymbol{\omega}} V^{\boldsymbol{\omega}_t}(s_t)}_{\widehat{\nabla}_{\boldsymbol{\omega}} L(\boldsymbol{\omega}_t)}, \tag{2.35}$$

where $\alpha_t > 0$ is a learning rate. This update is to be repeated for $t = 0, 1, \ldots, H-1$ and for any number of episodes. In practice, the updates are concentrated at the end of each episode, when the G_t can be actually computed. This *Monte Carlo* (MC) evaluation algorithm is just an instance of *Stochastic Gradient Descent* (SGD) with the MSE from (2.32) as a loss, since the update vector $\widehat{\nabla}_{\boldsymbol{\omega}} L$ in (2.35) is such that $\mathbb{E}_\pi[\widehat{\nabla}_{\boldsymbol{\omega}} L(\boldsymbol{\omega})] = \nabla_{\boldsymbol{\omega}} L(\boldsymbol{\omega})$. Convergence to a *local* minimum of (2.32) is guaranteed, under some regularity assumptions (Bottou, 1998, Section 5.1), if the sequence of learning rates satisfies Robbins-Monro's conditions (Robbins and Monro, 1951):

$$\sum_{t=0}^{\infty} \alpha_t = \infty \qquad \text{and} \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty. \tag{2.36}$$

This entails convergence to the *global* minimum for linear function approximation, for which the loss (2.32) is convex. However, the true value function can only be recovered if the approximation is realizable, for instance in tabular RL.

The main issue of MC approaches is the high variance of the updates, which is a cause of slow convergence. This variance is due to the long sum of random variables implicit in the definition of the return-to-go G_t, and is worse for longer episodes. A first way to reduce variance is to aggregate updates over N trajectories:

$$\widehat{\nabla}^N_{\boldsymbol{\omega}} L(\boldsymbol{\omega}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \left(G_t^{(i)} - Q^{\boldsymbol{\omega}}(s_t^{(i)}, a_t^{(i)})\right) \nabla_{\boldsymbol{\omega}} Q^{\boldsymbol{\omega}_k}(s_t^{(i)}, a_t^{(i)}), \tag{2.37}$$

where superscripts denote membership to the i-th trajectory. This, of course, produces even less frequent updates: just one every batch of N episodes.

2.7.2 Temporal-difference evaluation

When the interaction is not episodic, we can resort to *Temporal Difference* (TD) methods. Another reason to use TD is to reduce the variance of MC at the price of some bias, by truncating the sum of future rewards. This may yield faster convergence in practice. Finally, the updates of TD are truly online since we do not have to wait the end of the episode to compute them.

Temporal difference can be understood as an approximation to the iterative policy evaluation algorithm described in Section 2.4.1. The *Bellman Error* (BE) for state s under value-function parameters $\boldsymbol{\omega}$ is defined as the difference in predicted value after one application of Bellman's expectation operator:

$$BE(s; \boldsymbol{\omega}) = \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} \left[r(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim p(\cdot|s,a)} \left[V^{\boldsymbol{\omega}}(s') \right] \right] - V^{\boldsymbol{\omega}}(s). \tag{2.38}$$

By definition of V^π, this error would be uniformly zero if $V^{\boldsymbol{\omega}} = V^\pi$. So, we would like to refine $\boldsymbol{\omega}$ as to minimize the BE. Let $(s_t, a_t, s_{t+1}, r_{t+1})$ be the result of a single step of interaction between policy π and the environment. This kind of sample is sometimes called a *transition*. The *Temporal Difference* (TD) error can be computed as:

$$\delta_t^{\boldsymbol{\omega}}(s_t) = r_{t+1} + \gamma V^{\boldsymbol{\omega}}(s_{t+1}) - V^{\boldsymbol{\omega}}(s_t). \tag{2.39}$$

Note that $BE(s_t; \boldsymbol{\omega}) = \mathbb{E}_\pi \left[\delta_t^{\boldsymbol{\omega}}(s_t)\right]$. This suggests to update the value function estimate in the direction of decreasing squared TD error:

$$\boldsymbol{\omega}_{t+1} \leftarrow \boldsymbol{\omega}_t + \alpha_t \delta_t^{\boldsymbol{\omega}}(s_t) \nabla_{\boldsymbol{\omega}} V^{\boldsymbol{\omega}_t}(s_t), \tag{2.40}$$

where $\alpha_t > 0$ is a learning rate. Intuitively, we are (approximately) updating $V^{\boldsymbol{\omega}}$ in the direction of its *Bellman target* $T^\pi V^{\boldsymbol{\omega}}$. Compared to the MC approach, this is only a *semi-gradient* update, since the target is treated as a constant when, in fact, it also depends on $\boldsymbol{\omega}$. This is a form of *bootstrapping*. Moreover, the samples used to perform TD updates are not i.i.d. like the trajectories used in MC evaluation. Convergence is still guaranteed for *linear* function approximation and Robbins-Monro learning-rate sequences (Tsitsiklis and Van Roy, 1996), but not for more general approximations (like deep neural networks). Unfortunately, even in the linear case, the resulting value function is *not* the global minimizer of the MSE. Said ω_∞ the fixed point of the TD update (2.40), we can only guarantee that (Tsitsiklis and Van Roy, 1996):

$$L(\boldsymbol{\omega}_\infty) \leqslant \frac{1}{1-\gamma} \min_{\boldsymbol{\omega} \in \Omega} L(\boldsymbol{\omega}). \tag{2.41}$$

This bias, which is in addition to function-class bias (2.33), is the price to pay for the reduced variance of TD w.r.t. MC.

The algorithm outlined in this section is known as TD(0) (Sutton, 1988). Several improvements have been proposed in the RL literature. These include smarter ways of trading-off bias and variance, such as n-step returns and eligibility traces (Sutton and Barto, 2018; van Seijen et al., 2016), full-gradient methods (Sutton et al., 2008; Yu, 2017) and more data-efficient techniques such as LSTD (Least Squares Temporal Difference, Bradtke and Barto, 1996; Boyan, 2002).

2.7.3 Learning Q

The techniques presented in the previous sections can be extended to learn quality functions, which are often more useful in policy optimization. Consider a quality-function approximator $Q^{\boldsymbol{\omega}}$ parametrized by vector $\boldsymbol{\omega} \in \mathbb{R}^d$. The MC evaluation algorithm presented in Section 2.7.1 can be applied as is, since the return-to-go G_t experienced under π is also an unbiased estimate of $Q^\pi(s_t, a_t)$. As for TD, we just need to overload the definition of the TD error to also account for actions:

$$\delta_t^{\boldsymbol{\omega}}(s_t, a_t) = r_{t+1} + \gamma Q^{\boldsymbol{\omega}}(s_{t+1}, a_{t+1}) - Q^{\boldsymbol{\omega}}(s_t, a_t). \tag{2.42}$$

This requires samples of the kind $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$. If updates are performed entirely online, they need to be delayed by one control step (to observe a_{t+1}) compared to V-function estimation.

2.7.4 Off-policy policy evaluation

In off-policy policy evaluation we want to estimate Q^π (or V^π) for target policy π from samples collected with a different behavioral policy π_B.

The MC approach is readily adapted to the off-policy case by *Importance Sampling* (IS):

$$\widehat{\nabla}^N_{\boldsymbol{\omega}} L(\boldsymbol{\omega}) = \frac{1}{N}\sum_{i=1}^{N} w_{\pi/\pi_B}(\tau_i) \sum_{t=0}^{H} \left(G_t^{(i)} - Q^{\boldsymbol{\omega}}(s_t^{(i)}, a_t^{(i)})\right) \nabla_{\boldsymbol{\omega}} Q^{\boldsymbol{\omega}_k}(s_t^{(i)}, a_t^{(i)}), \quad (2.43)$$

where $w_{\pi/\pi_B}(\tau_i) = p_\pi(\tau_i)/p_{\pi_B}(\tau_i)$ is the *importance weight* for trajectory τ_i, sampled from p_{π_B}. This is only meaningful is the behavioral policy is stochastic and absolutely continuous w.r.t. π. An in-depth discussion of IS is provided in Section 6.1.1. However, note that:

- ◇ The importance weight for trajectories can be easily computed as a product of policy ratios:

$$w_{\pi/\pi_B}(\tau) = \prod_{t=0}^{H-1} \frac{\pi_B(a_t|s_t)}{\pi(a_t|s_t)}, \quad (2.44)$$

  since the transition probabilities are the same for the two trajectory distributions and cancel out.

- ◇ The IS gradient estimator (2.43) is unbiased, hence the convergence properties of on-policy MC evaluation are preserved.

- ◇ The variance of the IS estimator (with the same batch size N) can be much higher than the one of the on-policy MC estimator, more so if the two policies induce very different trajectory distributions or the time horizon is very long.

Importance sampling can also be applied to TD methods. As shown by Precup et al. (2000), a simple adjustment of the TD error:

$$\delta_t^{IW}(s_t, a_t) = r_{t+1} + \gamma \frac{\pi(a_t|s_t)}{\pi_B(a_t|s_t)} Q^{\boldsymbol{\omega}}(s_{t+1}, a_{t+1}) - Q^{\boldsymbol{\omega}}(s_t, a_t), \quad (2.45)$$

is enough to guarantee convergence *in the tabular case* (also in combination with eligibility traces). Unfortunately, divergence can happen even for linear function approximation (Baird III, 1995). For this reason, the combination of off-policy learning, bootstrapping and function approximation is known as the *deadly triad*.

Further approaches to off-policy policy evaluation include doubly-robust estimators (Dudík et al., 2011; Jiang and Li, 2016; Thomas and Brunskill, 2016; Farajtabar et al., 2018), state-occupancy-ratio estimation (Hallak and Mannor, 2017; Liu et al., 2018b; Gelada and Bellemare, 2019), emphatic weighting (Sutton et al., 2016), gradient-TD methods (Sutton et al., 2009), and conservative approaches (Thomas et al., 2015b).

2.8 Value-Based Control

In this section, we briefly review value-based control methods, as these are often compared and opposed to policy-based optimization algorithms. We just present the main algorithmic ideas: as for value based-prediction, refer to Sutton and Barto (2018) for a comprehensive treatment.

Since the actor-critic algorithms described in Section 3.7 learn both a policy and a value function, a clarification is due. We call *value-based* algorithms those that learn primarily a value function and obtain a policy as a byproduct.

The algorithms described in this section assume a finite action space. Function approximation is still needed to manage continuous states. Although extensions to continuous actions are sometimes possible (Bradtke, 1992), policy-based algorithms represent a more natural solution in that case.

2.8.1 SARSA

The RL analogue of the policy iteration algorithm (Section 2.4.2) is called *SARSA* from the $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ samples used to compute the TD error (2.42). SARSA alternates a step of TD evaluation for the Q-function:

$$\boldsymbol{\omega}_{t+1} \leftarrow \boldsymbol{\omega}_t + \alpha_t \delta_t^{\boldsymbol{\omega}}(s_t, a_t) \nabla_{\boldsymbol{\omega}} Q^{\boldsymbol{\omega}}(s_t, a_t), \tag{2.46}$$

with ϵ-greedy action selection:

$$a_t = \begin{cases} \arg\max_{a \in \mathcal{A}} Q^{\boldsymbol{\omega}}(s_t, a) \text{ with probability } 1 - \epsilon \\ a \sim \mathcal{U}(\mathcal{A}) \text{ with probability } \epsilon, \end{cases} \tag{2.47}$$

for some exploration parameter $\epsilon > 0$. The latter corresponds almost to a policy improvement step, but random actions are performed with non-zero probability to guarantee a sufficient amount of exploration. SARSA with linear function approximation and Robbins-Monro learning rate (2.36) guarantees that the Q-function parameter $\boldsymbol{\omega}_t$ converges to a bounded region almost surely (Gordon, 2000). Many refinements of TD(0), such as eligiblity traces, can also be applied to SARSA (van Seijen et al., 2016). *Expected SARSA* (John, 1994; van Seijen et al., 2009) removes an unnecessary source of variance by averaging over the next action in the evaluation step instead of relying on the sample a_{t+1}:

$$\boldsymbol{\omega}_{t+1} \leftarrow \boldsymbol{\omega}_t - \alpha_t \left(r_{t+1} + \gamma \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s_t)} [Q^{\boldsymbol{\omega}}(s_t, a)] - Q^{\boldsymbol{\omega}}(s_t, a_t) \right) \nabla_{\boldsymbol{\omega}} Q^{\boldsymbol{\omega}}(s_t, a_t). \tag{2.48}$$

The closed-form expectation only adds a negligible computational burden in the finite-action setting. The same update can be used in the off-policy case without the need of adding an importance weight. However, the convergence issues pointed out for off-policy TD with function approximation in Section 2.7.4 apply both to SARSA and Expected SARSA.

2.8.2 Q-learning

The RL analogue of value iteration (Section 2.4.3) is an off-policy algorithm called Q-learning (Watkins, 1989). It can be framed as an instance of off-policy Expected Sarsa where the target policy is a greedy one:

$$\boldsymbol{\omega}_{t+1} \leftarrow \boldsymbol{\omega}_t - \alpha_t \left(r_{t+1} + \gamma \max_{a \in \mathcal{A}} \{Q^{\boldsymbol{\omega}}(s_t, a)\} - Q^{\boldsymbol{\omega}}(s_t, a_t) \right) \nabla_{\boldsymbol{\omega}} Q^{\boldsymbol{\omega}}(s_t, a_t). \quad (2.49)$$

The behavioral policy, typically an ϵ-greedy one (2.47), is chosen to guarantee a sufficient amount of exploration. Q-learning with Robbins-Monro learning rate (2.36) converges in the tabular case (Watkins and Dayan, 1992; Jaakkola et al., 1994a; Tsitsiklis, 1994), also in combination with eligibility traces (Watkins, 1989; Peng and Williams, 1994; Munos et al., 2016).[12] Unfortunately, function approximation completes the deadly triad (Section 2.7.4) causing divergence issues (e.g., Bradtke, 1992).

Albeit the lack of convergence guarantees, Q-learning combined with deep neural networks has experienced extraordinary empirical success in recent times. The *Deep Q-Network* (DQN) algorithm (Mnih et al., 2015) learned to play several Atari 2600 video-games from visual input. Besides using convolutional neural networks to manage the visual input and other engineering devices, DQN employs two main tricks to stabilize the learning process and avoid divergence in practice:

1. *Experience replay*: (s, a, r, s') samples collected by interaction are stored in a buffer, from which they are randomly sampled to perform Q-function updates. This breaks some correlations in the contiguous updates of Q-learning that may cause divergence.

2. *Target networks*: the Bellman target is estimated from a copy $Q^{\boldsymbol{\omega}^-}$ of the Q-function network whose weights $\boldsymbol{\omega}^-$ are updated periodically:

$$\boldsymbol{\omega}_{t+1} \leftarrow \boldsymbol{\omega}_t - \alpha_t \left(r_{t+1} + \gamma \max_{a \in \mathcal{A}} \{Q^{\boldsymbol{\omega}^-}(s_t, a)\} - Q^{\boldsymbol{\omega}}(s_t, a_t) \right) \nabla_{\boldsymbol{\omega}} Q^{\boldsymbol{\omega}}(s_t, a_t).$$

   This reduces the correlation between the updated weights and the target, which may also be a source of instabilities. Intuitively, delaying the update of the target network has the effect of slowing down the "moving target" typical of bootstrapping approaches.

The impressive results of DQN motivated a long series of improvement. Double DQN (van Hasselt et al., 2016) reduces the overestimation bias of Q-learning (van Hasselt, 2010) by employing separate Q-networks for selecting the greedy action and estimating its value. Dueling DQN (Wang et al., 2016b) is an architectural improvement that decomposes the Q-function into the sum of a value and an advantage function as a form of inductive bias. The replay buffer can also be

[12] The convergence of Watkins' $Q(\lambda)$ was one of the most long-standing open problems in RL. Proposed by Watkins (1989), it was finally proved to converge by Munos et al. (2016) after 27 years.

improved by prioritizing transitions with higher TD error (Schaul et al., 2016). These and other refinements to DQN are summarized by Hessel et al. (2018) and combined to form the almighty *Rainbow* algorithm.

In the next chapter we will discuss the advantages of policy-based algorithms over DQN when moving from a discrete, controlled world (such as that of Atari games) to the noisy and continuous world of robots.

Policy Optimization 3

This chapter provides an overview of the principles and methods of policy-based RL for continuous control. The terms *Policy Optimization* (PO) and *Policy Search* (Deisenroth et al., 2013) will be used as synonyms, although we prefer the former for its link to mathematical optimization. Technically speaking, a PO algorithm is one that searches over the space of policies directly, without defining the target policy as a function of another learned object, such as a value function.

The structure of this chapter is as follows. We start by providing some motivation for preferring PO to other RL frameworks in Section 3.1. In Section 3.2, we discuss some features of the policy optimization problem that make it different than both value-based RL and mathematical optimization. In Section 3.3, we introduce *parametric policies*, which allow to define a computationally feasible optimization problem even for continuous MDPs, and describe the most common families of parametric policies. In Section 3.4 we review the main theory on *Policy Gradient* (PG) approaches, which represent the most popular family of policy optimization algorithms. The next sections are specifically on policy gradient algorithms: we discuss actor-only algorithms in Section 3.5, natural gradient approaches in Section 3.6, actor-critic algorithms in Section 3.7, trust-region methods in Section 3.8, and deterministic policy gradients in Section 3.9. In Section 3.10, we present some advanced policy gradient algorithms developed for large-scale problems. In Section 3.11, we discuss an exploration technique based on entropy regularization. Although our focus is on policy gradient algorithms, other policy optimization approaches has proven successful, especially in the field of robot learning. We review some of these approaches in Section 3.12. We conclude by mentioning some applications of policy optimization algorithms in Section 3.13.

As mentioned in Section 2.6, we neglect model-based algorithms, which are nonetheless important for real-world RL. Refer to Nguyen-Tuong and Peters (2011) and Deisenroth et al. (2013, Chapter 3) for surveys on the topic.

3.1 Motivation for Policy Optimization

A first methodological advantage of policy optimization over value-based approaches is that it makes easier to adapt the methods of supervised learning to RL. From a theoretical standpoint, convergence guarantees under function approximation are the main point in favor of PO. The practical advantages of PO are particularly evident when applied to real-world continuous control, especially in the presence of sensor and actuator noise, continuous actions, partial observability and safety concerns.

Convergence guarantees

Several PO algorithms have convergence guarantees even when the policy is represented by a non-linear function. In comparison, most value-based algorithms are only guaranteed to converge with linear value-function approximation. This allows to use complex policies explicitly designed for the given task or general purpose deep neural networks, which can be useful to process large inputs, with much less concern about divergent behavior.

Continuous actions

The value-based methods discussed in Section 2.8 involve a maximization over actions in order to compute greedy policies from learned value functions. This is impossible or computationally expensive for continuous action spaces. Policy-based algorithms avoid this problem by directly defining a policy function. When this is stochastic, it is often chosen so that actions can be sampled in a computationally efficient way.

Robustness to noise

The greedy (or ϵ-greedy) policies employed in value-based control are particularly sensitive to unexpected disturbances or drifts in the agent's sensors, which result in noisy or biased states. A small variation of the state may cause a very different action to be performed as a result of greedy selection, with possibly unexpected outcomes. In PO, policies can be designed to be robust to these phenomena.

Partial observability

Although we only consider fully observable environments in this manuscript, PO algorithms have some degree of robustness to partial observability. This is because, compared to value-based approaches, they rely less on the correctness of Bellman's equations. Moreover, PO algorithms allow to learn stochastic policies, which can be better than deterministic ones in partially observable MDPs (Singh et al., 1994).

Policy design: prior knowledge, safety and explainability

Policies can be designed with the desired level of human engineering, leaving to the PO only the fine-tuning of unknown parameters. This allows to incorporate a

great deal of prior domain knowledge into the learning agent from the start. In safety-critical domains, this allows to explicitly enforce some safety constraints, which may prove more challenging with indirectly learned greedy policies. It can also improve the explainability of the learned policy: when the initial one can be explained in human terms, so is the final one since it just the same controller with different parameters.

Drawbacks

Of course, PO also has its disadvantages. First of all, it does not (in general) produce a value function, which may be a useful device to make predictions or transfer knowledge to different tasks. Policy design can also overly restrict the set of feasible policies, ruling out unexpectedly good ones. Most importantly, convergence is typically guaranteed only to *local optima*. This makes PO more naturally suited to the fine-tuning of existing controllers rather than learning from scratch. However, exceptions exist both in theory (Section 7.7) and practice (Section 3.13).

3.2 Special Features of Policy Optimization

In this section, we discuss some peculiarities of Policy Optimization that make it at the same time different from classical RL and an interesting special case of mathematical optimization. Importantly, we also provide the formalization of the problem that will be used throughout this manuscript.

3.2.1 Scalar Objectives

Traditionally, the goal of RL *control* algorithms is to identify an optimal policy (see Section 2.3.1). As observed by Sutton and Barto (2018, Chapter 9), in continuous state spaces we cannot hope, in general, to identify the best action for each one of them. Hence, we use a distribution $\mu \in \Delta_{\mathcal{S}}$ to weight the relative importance of each state. This allows to introduce a scalar *performance* measure for policy π:

$$J_\mu(\pi) = \mathop{\mathbb{E}}_{s\sim\mu}\left[V^\pi(s)\right]. \tag{3.1}$$

This induces a total ordering on policies, i.e., π' is equivalent to or better than π if $J_\mu(\pi') \geqslant J_\mu(\pi)$. A natural choice for μ is the starting-state distribution. However, note that the starting state distribution used during training may differ from the one the agent will face after deployment, and the latter may provide a more meaningful performance measure.

For optimization, control, and safety purposes, we often consider restricted policy classes. Fixed a policy class $\Pi \subseteq \Delta_{\mathcal{A}}^{\mathcal{S}}$ and a state distribution $\mu \in \Delta_{\mathcal{S}}$, the problem of Policy Optimization is to identify a policy that maximizes the performance J_μ (3.1) within Π:

$$\pi^* \in \arg\max_{\pi\in\Pi} J_\mu(\pi). \tag{3.2}$$

This is a constrained optimization problem if Π is not the full set of policies $\Delta_{\mathcal{A}}^{\mathcal{S}}$.

3.2.2 (Non-)Convexity

In general, the performance J_μ is a non-convex function of policy π. As observed by Bhandari and Russo (2019), this also happens for unrestricted policy classes and is a consequence of the multi-step nature of the RL problem. This means (3.2) is a special case of non-convex optimization, which is NP-hard (Murty and Kabadi, 1987). As a consequence, policy optimization algorithms often resort to the easier task of identifying a *good* policy, that is one with sufficiently high performance for the given problem. For instance, Policy Gradient algorithms (Section 3.4 onwards) are guaranteed to find a *locally optimal policy*. Common non-convex optimization techniques, such as random initialization, can then be used to improve the quality of the solution.

However, policy optimization is still a *special* case of non-convex optimization, and its peculiarities can be exploited to design more efficient algorithms (Chapter 7) and even achieve global optimality (Section 7.7). Here we just mention the *hidden convexity* of the PO problem, highlighted, e.g., by Neu et al. (2017); Chu et al. (2019); Bhandari and Russo (2019); Efroni et al. (2020b); Zhang et al. (2021). By (3.1), (2.14) and the definitions of d_μ^π (2.15) and ν_μ^π (2.16):

$$\begin{aligned} J_\mu(\pi) = \int_{\mathcal{S}} \mu(s) V^\pi(s) \,\mathrm{d}s &= \frac{1}{1-\gamma} \int_{\mathcal{S}} d_\mu^\pi(s) \int_{\mathcal{A}} \pi(a|s) r(s,a) \,\mathrm{d}a \,\mathrm{d}s \\ &= \frac{1}{1-\gamma} \int_{\mathcal{S}} \int_{\mathcal{A}} \nu_\mu^\pi(s,a) r(s,a) \,\mathrm{d}a \,\mathrm{d}s \propto \langle \nu_\mu^\pi, r \rangle, \end{aligned} \tag{3.3}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product on $\mathcal{S} \times \mathcal{A}$. Written in this way, the performance is clearly a convex (in fact, linear) function of the state-action occupancy measure ν_μ^π. The non-convexity of (3.2), then, is all the in the potentially nonlinear relationship between policy π and its induced distribution ν_μ^π, which retains the complex, multi-step nature of the RL objective.[1] Writing the performance as in (3.3) also displays that no irregularity of the reward function (provided it is bounded) can disrupt the smoothness of the performance w.r.t. the policy, a fact that will prove fundamental for monotonic improvement (Chapter 5) and convergence (Chapter 7) guarantees. Restricting the policy class Π can further add to the non-convexity of the problem.

3.2.3 The Performance Difference Lemma

Framing RL as a mathematical optimization problem does not mean we are forgetting the underlying temporal structure. A key result by Kakade and Langford (2002, Lemma 6.1.) allows to write the performance difference of two policies as an expected advantage:

$$J_\mu(\pi') - J_\mu(\pi) = \int_{\mathcal{S}} \int_{\mathcal{A}} \nu_\mu^{\pi'}(s,a) A^\pi(s,a) \,\mathrm{d}a \,\mathrm{d}s. \tag{3.4}$$

[1] The O-REPS framework (Zimin and Neu, 2013) solves (3.2) directly for ν_μ^π using convex optimization tools. The challenge is then to find a policy π that induces the optimal occupancy.

This is known as the *Performance Difference Lemma* in the RL literature[2] and is ubiquitous in the theory of policy optimization.

3.3 Parametric Policies

Among restricted policy classes, of particular interest are *parametric policies*, which play a major role in policy optimization algorithms and are the main focus of this manuscript. Let $\Theta \subseteq \mathbb{R}^m$ be a parameter space for some $m \in \mathbb{N}$. A policy class parametrized by Θ is any $\Pi_\Theta = \left\{\pi_{\boldsymbol{\theta}} \in \Delta_{\mathcal{A}}^{\mathcal{S}} \mid \boldsymbol{\theta} \in \Theta\right\}$. The elements of Θ are real-valued, m-dimensional vectors called *policy parameters* and the elements of Π_Θ are parametric policies.

Notation When using parametric policies, we often abbreviate $\pi_{\boldsymbol{\theta}}$ as $\boldsymbol{\theta}$ in subscripts, superscripts and function arguments. For instance, $V^{\boldsymbol{\theta}}$ is short for $V^{\pi_{\boldsymbol{\theta}}}$, $d_\mu^{\boldsymbol{\theta}}$ is short for $d_\mu^{\pi_{\boldsymbol{\theta}}}$ and $p_{\boldsymbol{\theta}}$ is short for $p_{\pi_{\boldsymbol{\theta}}}$. Performance $J(\boldsymbol{\theta})$ can be short for $J(\pi_{\boldsymbol{\theta}})$, but more often denotes the mapping $\boldsymbol{\theta} \mapsto J(\pi_{\boldsymbol{\theta}})$.

3.3.1 Likelihood theory

Consider a class of *full-support differentiable* stochastic parametric policies Π_Θ:

$$\pi_{\boldsymbol{\theta}}(a|s) > 0 \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}, \boldsymbol{\theta} \in \Theta \text{ and} \tag{3.5}$$

$$\boldsymbol{\theta} \mapsto \pi_{\boldsymbol{\theta}}(a|s) \text{ is differentiable w.r.t. } \boldsymbol{\theta} \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}. \tag{3.6}$$

We call *score* the gradient of the log-likelihood with respect to the policy parameters, $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}} : \Theta \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}^m$. Intuitively, the score provides the direction in parameter space along which the likelihood (of an action in some state) is maximized. Fixed a state s and a $\boldsymbol{\theta} \in \Theta$, the expected value of the score under $\pi_{\boldsymbol{\theta}}$ is zero:

$$\begin{aligned}
\mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)\right] &= \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s)\, \mathrm{d}a \\
&= \nabla_{\boldsymbol{\theta}} \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s)\, \mathrm{d}a \\
&= \nabla_{\boldsymbol{\theta}} 1 = 0.
\end{aligned} \tag{3.7}$$

The covariance matrix of the score is called *Fisher Information*, since it measures the amount of information about the parameter $\boldsymbol{\theta}$ carried by an action sampled from $\pi_{\boldsymbol{\theta}}(\cdot|s)$:

$$F_s(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)^T\right]. \tag{3.8}$$

This *Fisher Information Matrix* (FIM) is a symmetric positive semidefinite $m \times m$ matrix (since it is a covariance matrix). The negative Hessian of the log-likelihood,

[2] This statement was already implicit in earlier works. For instance, see Proposition 1 by Burnetas and Katehakis (1997).

$-\nabla^2_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}$, is called *observed information*. The FIM can also be computed as the expected value of the observed information:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[-\nabla^2_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)\right] = -\mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\nabla_{\boldsymbol{\theta}}\left(\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)^T\right)\right] \\
&\quad = -\mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\nabla_{\boldsymbol{\theta}}\left(\frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a|s)^T}{\pi_{\boldsymbol{\theta}}(a|s)}\right)\right] \\
&\quad = -\mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\frac{\nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a|s)^T\pi_{\boldsymbol{\theta}}(a|s) - \nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a|s)\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a|s)^T}{\pi_{\boldsymbol{\theta}}(a|s)^2}\right] \\
&\quad = -\int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a|s)^T \,\mathrm{d}a + \mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)^T\right] \qquad (3.9) \\
&\quad = -\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)\right]^T + F_s(\boldsymbol{\theta}) \\
&\quad = F_s(\boldsymbol{\theta}), \qquad (3.10)
\end{aligned}$$

where (3.9) is from multiple applications of the *log trick* ($\nabla f = f\nabla \log f$) and (3.10) is from (3.7). This form is more convenient for implementation with automatic differentiation tools (see Section 3.6).

A positive definite FIM defines a Riemannian metric on the parameter space. This metric is invariant under sufficient statistics, that is, if we consider a different parametrization of the same policy class Π_Θ, the distances between action distributions, as measured by the Fisher metric, do not change. In fact, the Fisher metric is unique w.r.t. this property: any Riemannian metric that is invariant under sufficient statistics is the Fisher metric multiplied by a positive scalar. The Fisher information has a strong relationship with the *Kullback-Leibler divergence* (KL):

$$\begin{aligned}
\mathcal{KL}(\pi_{\boldsymbol{\theta}}(\cdot|s)\|\pi_{\boldsymbol{\theta}'}(\cdot|s)) &:= \mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\log \frac{\pi_{\boldsymbol{\theta}}(\cdot|s)}{\pi_{\boldsymbol{\theta}'}(\cdot|s)}\right] \\
&= \frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^T F_s(\boldsymbol{\theta})(\boldsymbol{\theta}' - \boldsymbol{\theta}) + o\left(\left\|\boldsymbol{\theta}' - \boldsymbol{\theta}\right\|^3\right). \qquad (3.11)
\end{aligned}$$

The equivalence is obtained by a simple Taylor expansion. The KL divergence is a common measure of dissimilarity between probability distributions, but it is not a metric, since it is does not satisfy the triangle inequality, and not even a *semimetric*, since it is not symmetric. It is, however, a Bregman divergence.

The definitions of score and Fisher information can be generalized to any parametric distribution. By taking the expectation of $F_s(\boldsymbol{\theta})$ under the (unnormalized) state-occupancy distribution of $\pi_{\boldsymbol{\theta}}$, we obtain the following:

$$F(\boldsymbol{\theta}) := \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s\sim d^{\boldsymbol{\theta}}_\mu}\left[F_s(\boldsymbol{\theta})\right]. \qquad (3.12)$$

As shown by Bagnell and Schneider (2003) and independently by Peters et al. (2003), this is precisely the FIM of the distribution induced by $\pi_{\boldsymbol{\theta}}$ over indefinite trajectories.[3] We provide a proof of this fact adapted to our framework:

[3]This fact is non-trivial and was not initially stated by S. M. Kakade, who first introduced the Fisher metric to the RL community. Nonetheless, Kakade already adopted (3.12) (in the *ergodic* formulation) for their Natural Policy Gradient algorithm (Kakade, 2001a).

Proof Let $p_{\pi_{\boldsymbol{\theta}},\gamma}$ be defined as in (2.21), abbreviated as $p_{\boldsymbol{\theta},\gamma}$ in the following. Its FIM is:

$$\begin{aligned} F(\boldsymbol{\theta}) &= -\mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta},\gamma}}\left[\nabla^2_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta},\gamma}(\tau)\right] \\ &= -\mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta},\gamma}}\left[\sum_{t=0}^{|\tau|-1} \nabla^2_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t)\right] \qquad (3.13) \\ &= -\mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta},\gamma}}\left[\sum_{t=0}^{|\tau|-1} \mathop{\mathbb{E}}_{a_t\sim\pi_{\boldsymbol{\theta}}(\cdot|s_t)}\left[\nabla^2_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t)\right]\right] \\ &= \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta},\gamma}}\left[\sum_{t=0}^{|\tau|-1} F_{s_t}(\boldsymbol{\theta})\right] \\ &= \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{s\sim d^{\pi}_{\mu}}\left[F_s(\boldsymbol{\theta})\right], \qquad (3.14) \end{aligned}$$

where (3.13) is from the properties of the logarithm and the fact that only the policy terms depend on $\boldsymbol{\theta}$, while the last equality is from Lemma 2.2. ■

3.3.2 Direct parametrization

In finite MDPs, we can explicitly assign a scalar parameter to each action probability, so that:

$$\pi_{\boldsymbol{\theta}}(a_i|s_j) = \boldsymbol{\theta}_{ij}, \qquad (3.15)$$

and $\boldsymbol{\theta} \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{A}|}$ is subject to the *simplex constraints*:

$$\boldsymbol{\theta}_{ij} \geqslant 0 \text{ for all } i \in [|\mathcal{S}|], j \in [|\mathcal{A}|] \qquad \text{and} \qquad \sum_{i=1}^{|\mathcal{A}|} \boldsymbol{\theta}_{ij} = 1 \text{ for all } i \in [|\mathcal{S}|]. \qquad (3.16)$$

The latter are necessary for the policy (3.15) to be well defined, and must be explicitly enforced by the learning algorithm. All stochastic and deterministic policies can be represented in this way. An issue of the direct parametrization is that $\pi_{\boldsymbol{\theta}+\delta}$ may fall outside the policy space even for an infinitesimal parameter update δ, by violating the simplex constraints.

3.3.3 Softmax policies

A common parametric policy class for finite action spaces (but arbitrary state spaces) is that of *Softmax* (or Gibbs, or Boltzmann) policies. Let $h : \Theta \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be an *energy function* and τ a positive scalar, called *temperature*. A softmax policy $\pi_{\boldsymbol{\theta}} \in \Pi_{\Theta}$ can be defined as:

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{\exp\left(h_{\boldsymbol{\theta}}(s,a)/\tau\right)}{\sum_{a'\in\mathcal{A}} \exp\left(h_{\boldsymbol{\theta}}(s,a')/\tau\right)}. \qquad (3.17)$$

Intuitively, the probability of an action is weighted by its energy, or *preference* $h_{\boldsymbol{\theta}}(\cdot|s)$. This can be any function parametrized by $\boldsymbol{\theta}$, such as a neural network. The temperature parameter can be set to regulate the amount of stochasticity of the distribution, with larger temperature corresponding to higher entropy.[4] The denominator in (3.17) is a normalization factor ensuring that $\pi_{\boldsymbol{\theta}}(\cdot|s)$ is a probability distribution. One can sample efficiently from a Softmax distribution by using the *Gumbel trick*:

$$a = \arg\max_{a' \in \mathcal{A}} \left\{ \frac{1}{\tau} h_{\boldsymbol{\theta}}(s, a') + \eta_{a'} \right\}, \tag{3.18}$$

where $(\eta_{a'})_{a' \in \mathcal{A}}$ are i.i.d. $Gumbel(0, 1)$ random variables.

The score of a Softmax policy is:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) &= \frac{1}{\tau} \nabla_{\boldsymbol{\theta}} h_{\boldsymbol{\theta}}(s, a) - \nabla_{\boldsymbol{\theta}} \log \sum_{a' \in \mathcal{A}} \exp\left(h_{\boldsymbol{\theta}}(s, a')/\tau\right) \\
&= \frac{1}{\tau} \nabla_{\boldsymbol{\theta}} h_{\boldsymbol{\theta}}(s, a) - \frac{\sum_{a' \in \mathcal{A}} \nabla_{\boldsymbol{\theta}} \exp\left(h_{\boldsymbol{\theta}}(s, a')/\tau\right)}{\sum_{a' \in \mathcal{A}} \exp\left(h_{\boldsymbol{\theta}}(s, a')/\tau\right)} \\
&= \frac{1}{\tau} \nabla_{\boldsymbol{\theta}} h_{\boldsymbol{\theta}}(s, a) - \sum_{a' \in \mathcal{A}} \frac{\exp\left(h_{\boldsymbol{\theta}}(s, a')/\tau\right)}{\sum_{a'' \in \mathcal{A}} \exp\left(h_{\boldsymbol{\theta}}(s, a'')/\tau\right)} \frac{1}{\tau} \nabla_{\boldsymbol{\theta}} h_{\boldsymbol{\theta}}(s, a') \\
&= \frac{1}{\tau} \left(\nabla_{\boldsymbol{\theta}} h_{\boldsymbol{\theta}}(s, a) - \mathop{\mathbb{E}}_{a' \sim \pi_{\boldsymbol{\theta}}(\cdot|s)} \left[\nabla_{\boldsymbol{\theta}} h_{\boldsymbol{\theta}}(s, a') \right] \right).
\end{aligned} \tag{3.19}$$

Linear Softmax

A simple, but powerful parametrization is the *linear* one:

$$h_{\boldsymbol{\theta}}(s, a) = \boldsymbol{\theta}^T \phi(s, a), \tag{3.20}$$

where $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^m$ is a *feature map*. In this case the score is simply:

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) = \frac{1}{\tau} \left(\phi(s, a) - \mathop{\mathbb{E}}_{a' \sim \pi_{\boldsymbol{\theta}}(\cdot|s)} \left[\phi(s, a') \right] \right), \tag{3.21}$$

and the observed information has a compact expression:

$$\begin{aligned}
-\nabla^2_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) &= \frac{1}{\tau} \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{a' \sim \pi_{\boldsymbol{\theta}}(\cdot|s)} \left[\phi(s, a')^T \right], \\
&= \frac{1}{\tau} \mathop{\mathbb{E}}_{a' \sim \pi_{\boldsymbol{\theta}}(\cdot|s)} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \phi(s, a')^T \right] \\
&= \frac{1}{\tau^2} \mathop{\mathbb{E}}_{a' \sim \pi_{\boldsymbol{\theta}}(\cdot|s)} \left[\phi(s, a') \phi(s, a')^T - \phi(s, a') \mathop{\mathbb{E}}_{a'' \sim \pi_{\boldsymbol{\theta}}(\cdot|s)} \left[\phi(s, a'')^T \right] \right].
\end{aligned} \tag{3.22}$$

This is also the FIM $F_s(\boldsymbol{\theta})$ since it does not depend on a.

[4] We chose to include the temperature τ as a constant hyper-parameter instead of a learnable parameter since it is redundant, provided that the energy function is sufficiently expressive.

Tabular Softmax

In the special case of finite actions *and states*, we can parametrize the Softmax policy as to represent *all* possible full-support stochastic policies. Some additional care is needed to make the parametrization non-redundant. Let $|\mathcal{S}| = n$ and $|\mathcal{A}| = d$. Since we no longer need to generalize over the state space, we can consider each state $s_j \in \mathcal{S}$ separately. Let $\boldsymbol{\theta}_i \in \mathbb{R}^{d-1}$ for $i = 1, \dots, n$, θ_{ij} denote the j-th element of $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta} = [\boldsymbol{\theta}_1^T|, \dots, |\boldsymbol{\theta}_n^T]^T$ be the concatenation of all the $\boldsymbol{\theta}_i$. A possible non-redundant parametrization is the following:

$$\pi_{\boldsymbol{\theta}}(a_j|s_i) = \begin{cases} \frac{\exp(\theta_{ij}/\tau)}{1+\sum_{k=1}^{d-1}\exp(\theta_{ik}/\tau)} & \text{if } j \in [d-1], \\ \frac{1}{1+\sum_{k=1}^{d-1}\exp(\theta_{ik}/\tau)} & \text{if } j = d, \end{cases} \tag{3.23}$$

As observed, e.g., by Bhandari and Russo (2019), this policy space contains all full-support policies. However, $\Pi_\Theta \subset \Delta_{\mathcal{A}}^{\mathcal{S}}$ since deterministic policies (or even policies assigning zero probability to some action in some state) are only approached in the limit of very large policy parameters (or very small temperature, were we to let it vary). In other words, Π_Θ is the *interior* of $\Delta_{\mathcal{A}}^{\mathcal{S}}$. Let $\pi_{\boldsymbol{\theta}}(\cdot|s)$ denote the $(d-1)$-th dimensional vector $[\pi_{\boldsymbol{\theta}}(a_j|s)]_{j=1}^{d-1}$. Here $\mathbf{e}_j$ is the j-th base vector of $\mathbb{R}^{d-1}$ and $\mathbf{e}_d = \mathbf{0}$. We compute gradients w.r.t. to $\boldsymbol{\theta}_i$ alone:

$$\nabla_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}}(a_j|s_i) = \frac{1}{\tau}\left(\mathbf{e}_j - \pi_{\boldsymbol{\theta}}(\cdot|s_i)\right), \tag{3.24}$$

$$-\nabla^2_{\boldsymbol{\theta}_i} \log \pi_{\boldsymbol{\theta}}(a_j|s_i) = \frac{1}{\tau^2}\left(\operatorname{diag}(\pi_{\boldsymbol{\theta}}(\cdot|s_i)) - \pi_{\boldsymbol{\theta}}(\cdot|s_i)\pi_{\boldsymbol{\theta}}(\cdot|s_i)^T\right), \tag{3.25}$$

where the latter is also $F_{s_i}(\boldsymbol{\theta}_i)$ since it does not depend on a_i. One can complete with zeros to obtain the same quantities w.r.t. the complete parameter $\boldsymbol{\theta}$, since $\boldsymbol{\theta}_i$ has no influence on the probability of actions in states other than s_i. As observed by Metelli et al. (2019), the FIM $F_{s_i}(\boldsymbol{\theta}_i)$ is *strictly diagonally dominant by rows*, hence positive definite. The non-redundant parametrization is crucial for this last consideration.

One can still apply the Gumbel trick to sample from $\pi_{\boldsymbol{\theta}}$, by noting that $h_{\boldsymbol{\theta}}(s_i, a_j) = \theta_{ij}$ and $h_{\boldsymbol{\theta}}(s_i, a_d) = 0$ for all $i \in [n]$ and $j \in [d-1]$.

3.3.4 Gaussian policies

When the action space is continuous, the most common parametric policy is the Gaussian. Let us first consider scalar actions, i.e., $\mathcal{A} = \mathbb{R}$. A Gaussian policy is defined as:

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{1}{\sqrt{2\pi}\sigma_{\boldsymbol{\theta}}(s)} \exp\left(-\frac{\left(a - \mu_{\boldsymbol{\theta}}(s)\right)^2}{2\sigma^2_{\boldsymbol{\theta}}(s)}\right), \tag{3.26}$$

where π with no subscript denotes the mathematical constant, $\sigma : \Theta \times \mathcal{S} \to (0, \infty)$ is the policy *standard deviation* function (σ^2 is the policy *variance*) and $\mu : \Theta \times \mathcal{S} \to \mathbb{R}$ is the policy *mean* function, for instance a neural network parametrized by $\boldsymbol{\theta}$ with the state as input. Differently from the Softmax policy, the amount of stochasticity

is entirely regulated by $\sigma_{\boldsymbol{\theta}}$. The class Π_Θ of Gaussian policies is always a restriction: for instance, it can never represent *multimodal* action distributions.

One can sample efficiently from a Gaussian policy as follows:

$$a = \mu_{\boldsymbol{\theta}}(s) + \sigma_{\boldsymbol{\theta}}(s)\eta, \tag{3.27}$$

where $\eta \sim \mathcal{N}(0,1)$ is a standard normal random variable.

The score of a Gaussian policy is:

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) = -\frac{\nabla_{\boldsymbol{\theta}}\sigma_{\boldsymbol{\theta}}(s)}{\sigma_{\boldsymbol{\theta}}(s)} + \frac{a-\mu_{\boldsymbol{\theta}}(s)}{\sigma^2_{\boldsymbol{\theta}}(s)}\left(\nabla_{\boldsymbol{\theta}}\mu_{\boldsymbol{\theta}}(s) + \frac{(a-\mu_{\boldsymbol{\theta}}(s))}{\sigma_{\boldsymbol{\theta}}(s)}\nabla_{\boldsymbol{\theta}}\sigma_{\boldsymbol{\theta}}(s)\right). \tag{3.28}$$

Shallow Gaussian

As a running example of continuous-action policy, we consider a Gaussian with linear mean and *homoscedastic* (i.e., state-independent) variance:

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{1}{\sqrt{2\pi}\sigma_{\boldsymbol{\theta}}} \exp\left(-\frac{\left(a-\boldsymbol{\theta}_\mu^T\phi(s)\right)^2}{2\sigma^2_{\boldsymbol{\theta}}}\right), \tag{3.29}$$

where $\phi : \mathcal{S} \to \mathbb{R}^m$ is a feature function, $\boldsymbol{\theta}_\mu \in \mathbb{R}^m$ is the vector of *mean parameters* ($\mu_{\boldsymbol{\theta}}(s) = \boldsymbol{\theta}_\mu^T\phi(s)$) and the policy standard deviation is parametrized by a separate parameter $\theta_\sigma \in \mathbb{R}$, with $\sigma_{\boldsymbol{\theta}} = \exp(\theta_\sigma) > 0$. The complete policy parameter vector is $\boldsymbol{\theta} = [\boldsymbol{\theta}_\mu^T|\theta_\sigma]^T \in \mathbb{R}^{m+1}$. Given the ubiquity of the policy in this manuscript, we call it simply the *shallow Gaussian*, as opposed to *deep* policies where the mean and/or the standard deviation are deep neural networks. We compute scores w.r.t. mean and variance parameters separately:

$$\nabla_{\boldsymbol{\theta}_\mu} \log \pi_{\boldsymbol{\theta}}(a|s) = \frac{a-\mu_{\boldsymbol{\theta}}(s)}{\sigma^2_{\boldsymbol{\theta}}}\phi(s), \tag{3.30}$$

$$\nabla_{\theta_\sigma} \log \pi_{\boldsymbol{\theta}}(a|s) = \frac{(a-\mu_{\boldsymbol{\theta}}(s))^2}{\sigma^2_{\boldsymbol{\theta}}} - 1. \tag{3.31}$$

We do the same for the observed information:

$$-\nabla^2_{\boldsymbol{\theta}_\mu} \log \pi_{\boldsymbol{\theta}}(a|s) = \frac{\phi(s)\phi(s)^T}{\sigma^2_{\boldsymbol{\theta}}}, \tag{3.32}$$

$$-\nabla^2_{\theta_\sigma} \log \pi_{\boldsymbol{\theta}}(a|s) = \frac{2\,(a-\mu_{\boldsymbol{\theta}}(s))^2}{\sigma^2_{\boldsymbol{\theta}}}, \tag{3.33}$$

$$-\nabla_{\boldsymbol{\theta}_\mu}\nabla_{\theta_\sigma} \log \pi_{\boldsymbol{\theta}}(a|s) = \frac{2\,(a-\mu_{\boldsymbol{\theta}}(s))}{\sigma^2_{\boldsymbol{\theta}}}\phi(s). \tag{3.34}$$

The FIM is then:

$$F_s(\boldsymbol{\theta}) = \left(\begin{array}{c|c} \frac{\phi(s)\phi(s)^T}{\sigma^2_{\boldsymbol{\theta}}} & \mathbf{0} \\ \hline \mathbf{0} & 2 \end{array}\right), \tag{3.35}$$

by noting that $\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[a-\mu_{\boldsymbol{\theta}}(s)\right] = 0$ and $\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[(a-\mu_{\boldsymbol{\theta}}(s))^2\right] = \sigma^2_{\boldsymbol{\theta}}$. This matrix is singular for $m > 1$, but $F(\boldsymbol{\theta}) = \mathbb{E}_{s\sim d^{\boldsymbol{\theta}}_\mu}\left[F_s(\boldsymbol{\theta})\right]$ can still be non-singular.

Multivariate Gaussian

Now consider a d-dimensional action space, $\mathcal{A} = \mathbb{R}^d$. We can use a multi-variate Gaussian distribution:

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_{\boldsymbol{\theta}}(s)|}} \exp\left(-\frac{1}{2}(a - \mu_{\boldsymbol{\theta}}(s))^T \Sigma_{\boldsymbol{\theta}}^{-1}(s)(a - \mu_{\boldsymbol{\theta}}(s))\right), \tag{3.36}$$

where $\mu : \Theta \times \mathcal{S} \to \mathbb{R}^d$ yields the d-dimensional mean, $\Sigma : \Theta \times \mathcal{S} \to \mathbb{R}^{d \times d}$ yields the *covariance matrix*, and $|\cdot|$ denotes the determinant.

A possible *linear* parametrization of the mean is the following:

$$\mu_{\boldsymbol{\theta}}(s) = \boldsymbol{\theta}_\mu \phi(s), \tag{3.37}$$

where $\phi : \mathcal{S} \to \mathbb{R}^m$ and $\boldsymbol{\theta}_\mu \in \mathbb{R}^{d \times m}$ is a matrix of mean parameters. A *diagonal* covariance matrix is typically used. A possible homoscedastic parametrization is:

$$\Sigma_{\boldsymbol{\theta}} = \text{diag}(\exp(2\boldsymbol{\theta}_\Sigma)), \tag{3.38}$$

where $\boldsymbol{\theta}_\Sigma \in \mathbb{R}^d$. With a diagonal covariance matrix, each action variable follows an independent Gaussian distribution. We say the policy is *factored*. Hence, all the results on the shallow Gaussian (3.29) trivially extend to its multivariate factored counterpart (3.37)(3.38).

If one wants to allow correlation between different action variables, a full (homoscedastic) covariance matrix can be defined in this way:

$$\Sigma_{\boldsymbol{\theta}} = L_{\boldsymbol{\theta}} L_{\boldsymbol{\theta}}^T, \tag{3.39}$$

where $L_{\boldsymbol{\theta}}$ is a lower triangular matrix with positive diagonal entries:

$$L_{\boldsymbol{\theta}} = \begin{bmatrix} \exp(\theta_{\Sigma 11}) & 0 & \cdots & 0 \\ \theta_{\Sigma 21} & \exp(\theta_{\Sigma 22}) & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ \theta_{\Sigma d1} & \theta_{\Sigma d2} & \cdots & \exp(\theta_{\Sigma dd}) \end{bmatrix}, \tag{3.40}$$

so that $\boldsymbol{\theta}_\Sigma$ has only $d(d-1)/2$ parameters and $\Sigma_{\boldsymbol{\theta}}$ is always positive definite. Moreover, actions can be sampled efficiently as follows:

$$a = \mu_{\boldsymbol{\theta}}(s) + L_{\boldsymbol{\theta}} \mathbf{z}, \tag{3.41}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$ is a vector of independent standard normal random variables.

Bounded support

Sometimes we need the agent's actions (consider scalar actions for simplicity) to lie within an interval $[a_{\min}, a_{\max}] \subset \mathbb{R}$, for instance because of the physical limitations of the controlled system, or for safety reasons. This is not directly supported by the Gaussian policy defined in (3.26), which has full support $\mathbb{R}$.

A possible workaround is to just clip actions before applying them:

$$\widetilde{a} = \max\left\{\min\left\{a, a_{\max}\right\}, a_{\min}\right\}, \tag{3.42}$$

and consider the clipping as part of the environment. This allows to use the full-support Gaussian policy as is. However, as observed by (Ciosek and Whiteson, 2020), this can slow down learning, by making out-of-range actions indistinguishable.

A *truncated Gaussian distribution* could be used to evenly redistribute the probability mass of the extreme actions within the interval $[a_{\min}, a_{\max}]$. Unfortunately, the truncated version of (3.26) becomes non-differentiable w.r.t. policy parameters. Fujita and Maeda (2018) propose a *clipped Gaussian* policy that concentrates the tail probability on the endpoints $a_{\min}$ and $a_{\max}$ instead. This results in multi-modal action distributions that can be learned with ease.

A more popular approach (used e.g., by Haarnoja et al., 2018) consists in applying a differentiable and invertible *squashing function* $f : \mathbb{R} \to [a_{\min}, a_{\max}]$ to the Gaussian action, for instance:

$$a = \frac{(a_{\max} - a_{\min})}{2} \tanh(u) + \frac{a_{\max} + a_{\min}}{2}, \tag{3.43}$$

where $u \sim \mathcal{N}(\mu_{\boldsymbol{\theta}}(s), \sigma^2)$. The *change of variable formula* can be used to derive the distribution of the squashed action, which is still differentiable w.r.t. policy parameters. This approach can still slow down learning as actions close to the boundaries have very similar effects, a manifestation of the well known *gradient saturation* problem.

Finally, we can use policy classes that naturally have a bounded support (see Section 3.3.6).

3.3.5 The exponential family

A generalization of both Gaussian and Softmax policies is the *exponential family*. A general exponential-family policy can be written as:

$$\pi_{\boldsymbol{\theta}}(a|s) = h(a; s) \exp\left(\eta(\boldsymbol{\theta}; s)^T T(a; s) - B(\boldsymbol{\theta}; s)\right), \tag{3.44}$$

where $\boldsymbol{\theta} \in \Theta$, $\eta : \Theta \times \mathcal{S} \to \mathbb{R}^d$, $T : \mathcal{A} \times \mathcal{S} \to \mathbb{R}^d$, $B : \Theta \times \mathcal{S} \to \mathbb{R}$ and $h : \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. The policy is in *canonical form* if $\eta \equiv \boldsymbol{\theta}$.

The linear Softmax policy (3.20) can be immediately written in canonical form as:

$$\begin{aligned} &\eta(\boldsymbol{\theta}; s) = \boldsymbol{\theta}, && T(a; s) = \frac{\phi(s, a)}{\tau}, \\ &B(\boldsymbol{\theta}; s) = \log \sum_{a' \in \mathcal{A}} \exp\left(\boldsymbol{\theta}^T \phi(s, a')\right), && h(a; s) = 1. \end{aligned} \tag{3.45}$$

The shallow Gaussian policy is recovered as:

$$\begin{aligned} &\eta(\boldsymbol{\theta}; s) = \left[\begin{array}{c|c} \frac{\boldsymbol{\theta}^T}{\sigma^2_{\boldsymbol{\theta}}} & -\frac{1}{2\sigma^2_{\boldsymbol{\theta}}} \end{array}\right]^T, && T(a; s) = \left[\begin{array}{c|c} a\phi(s)^T & a^2 \end{array}\right]^T, \\ &B(\boldsymbol{\theta}; s) = \frac{\left(\boldsymbol{\theta}^T \phi(s)\right)^2}{2\sigma^2_{\boldsymbol{\theta}}} + \log|\sigma^2_{\boldsymbol{\theta}}|, && h(a; s) = \frac{1}{\sqrt{2\pi}}. \end{aligned} \tag{3.46}$$

Expressing this policy in canonical form requires a reparametrization:

$$\boldsymbol{\eta}_1 = \frac{\boldsymbol{\theta}}{\sigma_{\boldsymbol{\theta}}^2}, \qquad \eta_2 = -\frac{1}{2\sigma_{\boldsymbol{\theta}}^2}, \qquad B(\boldsymbol{\eta}; s) = -\frac{\left(\boldsymbol{\eta}_1^T \phi(s)\right)^2}{4\eta_2^2} + \frac{1}{2}\log\left|\frac{1}{2\eta_2}\right|. \tag{3.47}$$

This is sometimes called *natural parametrization*, and is seldom used in practice. Although it represents the same class of policies as (3.26), the optimization landscape, hence the learning behavior, can be different (Pajarinen et al., 2019).

In general, the score of an exponential-family policy is:

$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) = \mathcal{J}_{\boldsymbol{\theta}} \eta(\boldsymbol{\theta}; s) T(a; s) - \nabla_{\boldsymbol{\theta}} B(\boldsymbol{\theta}; s) =_{\text{c.f.}} T(a; s) - \nabla_{\boldsymbol{\theta}} B(\boldsymbol{\theta}; s), \tag{3.48}$$

where the second equality holds only in canonical form (c.f.), and the FIM:

$$F_s(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}^2 B(\boldsymbol{\theta}; s) - \mathcal{J}_{\boldsymbol{\theta}}^2 \eta(\boldsymbol{\theta}; s) \mathop{\mathbb{E}}_{a \sim \pi_{\boldsymbol{\theta}}(\cdot|s)} [T(a; s)] =_{\text{c.f.}} \nabla_{\boldsymbol{\theta}}^2 B(\boldsymbol{\theta}; s). \tag{3.49}$$

3.3.6 Other parametric policies

Chou et al. (2017) suggest to use Beta policies (another member of the exponential family) for bounded action spaces:

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{\Gamma(\alpha_{\boldsymbol{\theta}}(s) + \beta_{\boldsymbol{\theta}}(s)}{\Gamma(\alpha_{\boldsymbol{\theta}}()s)\Gamma(\beta_{\boldsymbol{\theta}}(s))} a^{\alpha_{\boldsymbol{\theta}}(s)-1}(1-a)^{\beta_{\boldsymbol{\theta}}(s)-1}, \tag{3.50}$$

where $\alpha, \beta : \Theta \times \mathcal{S} \to (1, \infty)$ and Γ is the Gamma function. This policy has support $[0, 1]$, hence represents a natural solution to the issues of the Gaussian policy discussed in Section 3.3.4 (Bounded support).

More complex policies can be designed for specific tasks. By incorporating domain knowledge, this can speed-up the learning process, reducing it to the *fine-tuning* of a relatively small set of parameters. A classical example is that of *motor primitives* for robotics (Peters and Schaal, 2008b).

Deterministic and non-differentiable parametric policies are also used in practice. For instance, *rule-based policies* offer a great degree of human interpretability (Likmeta et al., 2020).

3.4 The Policy Gradient Theorem

Parametric policies allow to solve the PO problem (3.2) via *gradient ascent*. This approach has two main advantages: it comes with convergence guarantees (also for nonlinear policy parametrizations) and it allows to employ many powerful stochastic optimization tools originally developed for supervised learning, such as ADAM (Kingma and Ba, 2015). The Policy Gradient Theorem of Sutton et al. (1999) provides a convenient expression for the gradient of the performance w.r.t. policy parameters:

Theorem 3.1 (Policy Gradient, Sutton et al. (1999), Theorem 1) *For any differentiable parametric full-support policy $\pi_{\boldsymbol{\theta}}$:*

$$\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) = \frac{1}{1-\gamma} \int_{\mathcal{S}} d_\mu^{\pi_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s, a) \,\mathrm{d}a \,\mathrm{d}s.$$

We provide an alternative proof based on Lemma 2.1:

Proof First, we establish the following relationship between the gradient of the quality function and the one of the value function:

$$\nabla Q^{\boldsymbol{\theta}}(s,a) = \nabla \left(r(s,a) + \gamma \mathop{\mathbb{E}}_{s' \sim p(\cdot|s,a)} \left[V^{\boldsymbol{\theta}}(s') \right] \right) \tag{3.51}$$

$$= \gamma \mathop{\mathbb{E}}_{s' \sim p(\cdot|s,a)} \left[\nabla V^{\boldsymbol{\theta}}(s') \right], \tag{3.52}$$

where (3.51) is by definition (2.4). Then, we compute the gradient of the value function:

$$\begin{aligned}
\nabla V^{\boldsymbol{\theta}}(s) &= \nabla \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a) \,\mathrm{d}a \\
&= \int_{\mathcal{A}} \nabla \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a) + \pi_{\boldsymbol{\theta}}(a|s) \nabla Q^{\boldsymbol{\theta}}(s,a) \,\mathrm{d}a \\
&= \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \left(\nabla \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a) + \nabla Q^{\boldsymbol{\theta}}(s,a) \right) \mathrm{d}a \qquad (3.53) \\
&= \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \Big(\nabla \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a) \\
&\qquad + \gamma \int_{\mathcal{S}} p(s'|s,a) \nabla V^{\boldsymbol{\theta}}(s') \,\mathrm{d}s' \Big) \mathrm{d}a \qquad (3.54) \\
&= \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \nabla \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a) \,\mathrm{d}a + \gamma \int_{\mathcal{S}} p_{\boldsymbol{\theta}}(s'|s) \nabla V^{\boldsymbol{\theta}}(s') \,\mathrm{d}s' \\
&= \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \nabla \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a) \,\mathrm{d}a \\
&\qquad + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} d_s^{\boldsymbol{\theta}}(s') \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s') \nabla \log \pi_{\boldsymbol{\theta}}(a|s') Q^{\boldsymbol{\theta}}(s,a) \,\mathrm{d}a \,\mathrm{d}s', \qquad (3.55)
\end{aligned}$$

where (3.53) is from the *log trick* and requires the policy to have full support, (3.54) is from (3.52), and the last inequality is from Lemma 2.1 with $f = s \mapsto V^{\boldsymbol{\theta}}$ and $g = s \mapsto \int_{\mathcal{A}} \nabla \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a) \,\mathrm{d}a$. The proof is completed by taking the expectation of (3.55) under μ, by definition of J_μ (3.1) and d_μ (2.15). ■

By using (3.7), we can generalize the Policy Gradient Theorem to:

$$\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) = \frac{1}{1-\gamma} \int_{\mathcal{S}} d_\mu^{\pi_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \left(Q^{\pi_{\boldsymbol{\theta}}}(s,a) - b(s) \right) \mathrm{d}a \,\mathrm{d}s, \tag{3.56}$$

where $b : \mathcal{S} \to \mathbb{R}$ is any integrable function that does not depend on actions, called a *baseline*. This yields the following alternative expressions for the policy gradient:

$$\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) = \frac{1}{1-\gamma} \int_{\mathcal{S}} d_\mu^{\pi_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \left(Q^{\pi_{\boldsymbol{\theta}}}(s,a) - V^{\pi_{\boldsymbol{\theta}}}(s)\right) \mathrm{d}a \, \mathrm{d}s \tag{3.57}$$

$$= \frac{1}{1-\gamma} \int_{\mathcal{S}} d_\mu^{\pi_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) A^{\pi_{\boldsymbol{\theta}}}(s,a) \, \mathrm{d}a \, \mathrm{d}s \tag{3.58}$$

$$= \frac{1}{1-\gamma} \int_{\mathcal{S}} \int_{\mathcal{A}} \nu_\mu^{\pi_{\boldsymbol{\theta}}}(s,a) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) A^{\pi_{\boldsymbol{\theta}}}(s,a) \, \mathrm{d}a \, \mathrm{d}s. \tag{3.59}$$

The power of the Policy Gradient Theorem lies in the fact that the only first-order term we need to compute is the score function, which is either known in closed form or can be easily obtained via backpropagation with automatic differentiation tools.

3.4.1 Monte Carlo PGT

The first application of the theorem is an algorithm proposed by Sutton et al. (1999), known simply as *Policy Gradient Theorem* (PGT). To avoid any confusion, we will call it *Monte Carlo PGT*. Assume episodic interaction with indefinite-length trajectories, undiscounted rewards and continuation probability $\gamma < 1$. As mentioned in Section 2.7.3, the return-to-go G_t observed under $\pi_{\boldsymbol{\theta}}$ is an unbiased estimate of $Q^{\boldsymbol{\theta}}(s_t, a_t)$. This allows to build the following gradient estimator:

$$\widehat{\nabla}_\gamma J(\boldsymbol{\theta}) = \sum_{t=0}^{|\tau|-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) G_t, \tag{3.60}$$

which is unbiased:

$$\begin{aligned}
\mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}, \gamma} \left[\widehat{\nabla}_\gamma J(\boldsymbol{\theta})\right] &= (1-\gamma) \sum_{H=1}^{\infty} \gamma^{H-1} \sum_{t=0}^{H-1} \mathop{\mathbb{E}}_{\tau_{0:t} \sim p_{\boldsymbol{\theta}}} \left[\nabla \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \mathop{\mathbb{E}}_{\tau_{t+1:H-1} \sim p_{\boldsymbol{\theta}}} [G_t]\right] \\
&= (1-\gamma) \sum_{H=0}^{\infty} \gamma^H \sum_{t=0}^{H} \gamma^t \mathop{\mathbb{E}}_{\tau_{0:t} \sim p_{\boldsymbol{\theta}}} \left[\nabla \log \pi_{\boldsymbol{\theta}}(a_t|s_t) Q^{\boldsymbol{\theta}}(s_t, a_t)\right] \\
&= \sum_{t=0}^{\infty} \gamma^t \mathop{\mathbb{E}}_{\tau_{0:t} \sim p_{\boldsymbol{\theta}}} \left[\nabla \log \pi_{\boldsymbol{\theta}}(a_t|s_t) Q^{\boldsymbol{\theta}}(s_t, a_t)\right] \qquad (3.61) \\
&= \frac{1}{1-\gamma} \int_{\mathcal{S}} \int_{\mathcal{A}} \nu_\mu^{\boldsymbol{\theta}}(s,a) \nabla \log \pi_{\boldsymbol{\theta}}(s,a) Q^{\boldsymbol{\theta}}(s,a) \, \mathrm{d}a \, \mathrm{d}s, \qquad (3.62)
\end{aligned}$$

where (3.61) is from (2.26) and the last equality is from Lemma 2.2.

Monte Carlo PGT (Algorithm 1) uses (3.60) to update policy parameters in the direction of estimated performance improvement. It performs one policy update per episode.

Monte Carlo PGT is an instance of SGD. Sufficient conditions for the convergence to a local optimum, in the sense $\lim_{t\to\infty} \nabla J(\boldsymbol{\theta}) = 0$, are (Sutton et al., 1999, Theorem 3):

Algorithm 1 Monte Carlo PGT (Policy Gradient Theorem)

1: **Input:** initial policy parameters $\boldsymbol{\theta}_0$, step size sequence $(\alpha_k)_{k=0}^{\infty}$
2: **for** $k = 0, 1, 2 \dots$ **do**
3: Generate trajectory $\tau \sim p_{\boldsymbol{\theta},\gamma}$
4: Let $H = |\tau|$
5: **for** $t = 0, \dots, H-1$ **do**
6: $G_t = \sum_{i=t}^{H-1} \gamma^{i-t} r_{i+1}$
7: **end for**
8: $\widehat{\nabla}_{\gamma} J(\boldsymbol{\theta}_k) = \sum_{t=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}_k}(a_t|s_t) G_t$
9: $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha_k \widehat{\nabla}_{\gamma} J(\boldsymbol{\theta}_k)$
10: **end for**

1. The sequence $(\alpha_k)_{k=0}^{\infty}$ of step sizes is such that $\lim_{t\to\infty} \alpha_k = 0$ and $\sum_{k=0}^{\infty} \alpha_k = \infty$;

2. The MDP has bounded rewards, i.e., $\|r\|_{\infty} < \infty$;

3. $\sup_{s\in\mathcal{S}, a\in\mathcal{A}, \boldsymbol{\theta}\in\Theta} \left\|\nabla^2_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s)\right\|_{\infty} < \infty$.

We are not entirely satisfied with the third one. In fact, it already fails for the shallow Gaussian (3.29), since:

$$\nabla^2_{\boldsymbol{\theta}_\mu} \pi_{\boldsymbol{\theta}}(a|s) = \pi_{\boldsymbol{\theta}}(a|s) \left(\frac{a - \boldsymbol{\theta}_\mu^T \phi(s)}{e^{2\theta_\sigma}} - 1 \right) \frac{\phi(s)\phi(s)^T}{e^{2\theta_\sigma}}, \tag{3.63}$$

can take arbitrarily large values even if policy parameters *and* features are bounded, since $a \in \mathbb{R}$. One should enforce a bounded support, with all the issues discussed in Section 3.3.4. As already observed by Sutton et al. (1999), we just need the Hessian of the performance $\nabla^2 J(\boldsymbol{\theta})$ to be uniformly bounded (Bertsekas and Tsitsiklis, 1996). In Chapter 5 we provide weaker sufficient conditions for this property, and revisit this convergence guarantee in Section 7.2.1.

A remark on discounting

In presenting Monte Carlo PGT, we have considered indefinite-length trajectories and interpreted γ as a continuation probability, as this allows to reconcile Theorem 3.1 with Monte Carlo value estimation. In the finite-horizon, discounted-reward setting, the policy gradient estimate used in Line 8 of Algorithm 1 must be modified as follows:

$$\widehat{\nabla}_{H} J(\boldsymbol{\theta}) = \sum_{t=0}^{H-1} \gamma^t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) G_t, \tag{3.64}$$

by explicitly including the discount factor (cf. Sutton and Barto, 2018, Section 13.3). Indeed, under Assumption 2.1:

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s,a\sim\nu_\mu^{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\right] \\
&= \sum_{t=0}^{\infty} \gamma^t \mathop{\mathbb{E}}_{\tau_{0:t}\sim p_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) Q^{\boldsymbol{\theta}}(s_t,a_t)\right] \qquad (3.65) \\
&= \sum_{t=0}^{\infty} \gamma^t \mathop{\mathbb{E}}_{\tau_{0:t}\sim p_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \sum_{h=t}^{\infty} \gamma^{h-t} \mathop{\mathbb{E}}_{\tau_{t+1:h}\sim p_{\boldsymbol{\theta}}} \left[r_{h+1}\right]\right] \\
&= \sum_{t=0}^{H-1} \gamma^t \mathop{\mathbb{E}}_{\tau_{0:t}\sim p_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \mathop{\mathbb{E}}_{\tau_{t+1:H-1}\sim p_{\boldsymbol{\theta}}} \left[G_t\right]\right] \qquad (3.66) \\
&= \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta},H}} \left[\widehat{\nabla}_H J(\boldsymbol{\theta})\right],
\end{aligned}$$

where (3.65) is from Lemma 2.2 and (3.66) from Assumption 2.1.

This is not a matter of pedantry. Using (3.60) in the finite-horizon discounted-reward setting would be a serious mistake: as pointed out by Nota and Thomas (2020), $\mathbb{E}_{\tau\sim p_{\boldsymbol{\theta},H}}\left[\widehat{\nabla}_\gamma J(\boldsymbol{\theta})\right]$ is not the gradient of any function, and can even lead to policies that are *pessimal* both w.r.t. the discounted and undiscounted reward signal.

3.5 Actor-Only Policy Gradient Algorithms

We gather under the umbrella term *Policy Gradient* (PG) all algorithms that optimize parametric policies using first-order information. The Monte Carlo PGT from the previous section is a first example of PG. In particular, it is an *actor-only* PG algorithm, the actor being the parametric policy, as opposed to *actor-critic* algorithms (Section 3.7) that also learn a value function. The same rationale, applied to the finite-horizon setting, yields a family of PG algorithms generally identified by the name REINFORCE (original typesetting by Williams, 1992).[5] Our presentation of the topic is mainly based on (Peters and Schaal, 2008b).

The general scheme of an actor-only PG algorithm is reported in Algorithm 2. The current policy is updated once a batch of $N \in \mathbb{N}$ complete trajectories has been collected. For this reason, N is called the *batch size*. The dataset $\mathcal{D}_k = \{\tau_1, \ldots, \tau_N\}$ is the collection of said trajectories, where $\tau_1, \ldots, \tau_N \overset{\text{iid}}{\sim} p_{\boldsymbol{\theta}_k}$. In more compact notation, $\mathcal{D}_k \sim p_{\boldsymbol{\theta}_k}$. Provided $\widehat{\nabla} J$ is an unbiased estimator of the policy gradient, i.e., $\mathbb{E}_{\mathcal{D}\sim p_{\boldsymbol{\theta}}}\left[\widehat{\nabla} J(\boldsymbol{\theta};\mathcal{D})\right] = \nabla J(\boldsymbol{\theta})$, SGD theory can be used to guarantee convergence to a local optimum, under appropriate conditions on the sequence of step sizes and

[5] Using REINFORCE to name different algorithms—such as William's REINFORCE, G(PO)MDP (Baxter and Bartlett, 2001), Sutton's PGT (Sutton et al., 1999), and even more sophisticated actor-only policy gradient algorithms—is a common practice. However, it can be a source of confusion for students, researchers, and practitioners.

the regularity of $J(\boldsymbol{\theta})$ (Bottou, 1998). We discuss the convergence of actor-only PG algorithms in more depth in Chapter 7.

Specific algorithms, discussed in the following, differ mostly in how the policy gradient $\nabla_{\boldsymbol{\theta}} J$ is estimated. An important property of PG estimators is the variance, defined for any vector $\boldsymbol{v}$ of random variables as:

$$\mathbb{V}\text{ar}(\boldsymbol{v}) = \text{Tr}(\mathbb{C}\text{ov}(\boldsymbol{v}, \boldsymbol{v})) = \mathbb{E}\left[\left\|\boldsymbol{v} - \mathbb{E}\left[\boldsymbol{v}\right]\right\|^2\right], \tag{3.67}$$

which in the case of an unbiased PG estimator $\widehat{\nabla} J$ is:

$$\mathbb{V}\text{ar}(\widehat{\nabla} J(\boldsymbol{\theta})) = \mathop{\mathbb{E}}_{\mathcal{D}\sim p_{\boldsymbol{\theta}}}\left[\left\|\widehat{\nabla} J(\boldsymbol{\theta}; \mathcal{D}) - \nabla J(\boldsymbol{\theta})\right\|^2\right]. \tag{3.68}$$

The high variance of PG estimators is a known source of slow convergence (more in Chapter 7) and also makes safe learning difficult (Chapter 5).

Possible generalizations of the simple scheme outlined in Algorithm 2 include:

- ◇ Using a *vector step size* $\boldsymbol{\alpha}$ (Schraudolph et al., 2005; Papini et al., 2017), i.e., $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \boldsymbol{\alpha} \circ \nabla J(\boldsymbol{\theta}_k)$, where $\boldsymbol{\alpha}$ is a positive vector $[\alpha_1, \ldots, \alpha_m]^T$ and $\circ$ denotes the Hadamard (element-wise) product.
- ◇ Making the step size *adaptive*, i.e., iteration and/or data-dependent (Pirotta et al., 2013a). This includes the powerful adaptive-gradient techniques commonly used in supervised learning such as RMSPROP (Tieleman and Hinton, 2012) and ADAM (Kingma and Ba, 2015).
- ◇ Making the batch size N also adaptive (Papini et al., 2017). We discuss adaptive step *and* batch sizes in Chapter 5 for safety purposes.
- ◇ Applying a preconditioning matrix $G(\boldsymbol{\theta})$ to the gradient, as in Natural Policy Gradient (Kakade, 2001a), discussed in Section 3.6, and second-order methods (Furmston and Barber, 2012).[6]
- ◇ Re-using the same trajectories for multiple updates to increase data efficiency, as discussed in Chapter 6.

Algorithm 2 Actor-Only Policy Gradient

1: **Input:** initial policy parameters $\boldsymbol{\theta}_0$, step size sequence $(\alpha)_{k=0}^{\infty}$, batch size N
2: **for** $k = 0, 1 \ldots$ **do**
3: Collect N trajectories with $\boldsymbol{\theta}_k$ to obtain dataset $\mathcal{D}_k$
4: Compute policy gradient estimate $\widehat{\nabla} J(\boldsymbol{\theta}_k; \mathcal{D}_k)$
5: Update policy parameters as $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha_k \widehat{\nabla} J(\boldsymbol{\theta}_k; \mathcal{D}_k)$
6: **end for**

[6] Note that a vector step size $\boldsymbol{\alpha}$ can be seen as a special diagonal preconditioning matrix.

3.5.1 William's REINFORCE

The original REINFORCE algorithm (Williams, 1992) predates the PGT and can be derived without it. Assume episodic interaction with time-horizon H and discount factor $\gamma \in [0, 1]$. The performance of policy $\pi_{\boldsymbol{\theta}}$ can simply written as:

$$J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} [R(\tau)], \tag{3.69}$$

where R is the (discounted) return defined in (2.29). To see that this is indeed equivalent to (3.1) when $\gamma < 1$:

$$\begin{aligned}
J(\boldsymbol{\theta}) &= \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{s,a \sim \nu_\mu^{\boldsymbol{\theta}}} [r(s,a)] \\
&= \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta},\gamma}} \left[\sum_{t=0}^{|\tau|-1} r_{t+1} \right] && (3.70) \\
&= (1-\gamma) \sum_{T=1}^{\infty} \gamma^{T-1} \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta},T}} \left[\sum_{t=0}^{T-1} r_{t+1} \right] \\
&= (1-\gamma) \sum_{T=1}^{\infty} \gamma^{T-1} \sum_{t=0}^{T-1} \mathop{\mathbb{E}}_{\tau_{0:t} \sim p_{\boldsymbol{\theta}}} [r(s_t, a_t)] \\
&= \sum_{t=0}^{\infty} \gamma^t \mathop{\mathbb{E}}_{\tau_{0:t} \sim p_{\boldsymbol{\theta}}} [r(s_t, a_t)] && (3.71) \\
&= \sum_{t=0}^{H-1} \gamma^t \mathop{\mathbb{E}}_{\tau_{0:t} \sim p_{\boldsymbol{\theta}}} [r(s_t, a_t)] && (3.72) \\
&= \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}, H} [R(\tau)], && (3.73)
\end{aligned}$$

where (3.70) is from Lemma 2.2, (3.71) is from (2.26) and (3.72) is from Assumption 2.1.

The above equivalence (3.73) is intuitive, if not trivial. We stress its importance as it allows to transfer all the theory developed for the indefinite-horizon case to the finite-horizon setting under Assumption 2.1 on the time horizon and provided $\gamma < 1$. It justifies the approach, adopted several times in this manuscript, of developing general theory in the indefinite-horizon framework (with access to a first-order oracle), then moving to the finite-horizon setting to analyze the properties of practical gradient estimators.

Given (3.69), the gradient of the performance w.r.t. policy parameters is:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \int_{\mathcal{T}} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) R(\tau) \, \mathrm{d}\tau \\
&= \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} [\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) R(\tau)] && (3.74) \\
&= \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) R(\tau) \right], && (3.75)
\end{aligned}$$

where $\mathcal{T}$ denotes the set of all possible trajectories[7], (3.74) is from the log trick, and the last equality is from the properties of the logarithm and the fact that only action probabilities depend on policy parameters. This can be seen as the finite-horizon equivalent of the Policy Gradient Theorem (3.1).

The REINFORCE gradient estimator is just the Monte Carlo version of (3.75), which is clearly unbiased. Given a dataset $\mathcal{D} = \{\tau_1, \dots, \tau_N\}$ of N i.i.d. trajectories:

$$\widehat{\nabla}_R J(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t^{(i)} | s_t^{(i)}) \right) \left(\sum_{t=0}^{H-1} \gamma^t r_{t+1}^{(i)} \right), \tag{3.76}$$

where superscripts denote trajectory membership. As in Monte Carlo PGT (Algorithm 1), the only first-order terms we need to compute are the scores. The REINFORCE algorithm is just Algorithm 2 with (3.76) as the gradient estimator.

A remark on automatic differentiation The REINFORCE estimator admits a particularly efficient implementation with automatic differentiation tools, since it can be written as the gradient of a dot product between a vector of differentiable functions (the N sums of log-policies) and a vector of constants (the N returns).

3.5.2 G(PO)MDP

The REINFORCE algorithm does not fully exploit the temporal relationships between the terms in (3.76). The G(PO)MDP estimator (Baxter and Bartlett, 2001) (sometimes called GPOMDP or GMDP) is a refinement of REINFORCE that exploits the independence between rewards and *future* decisions[8]:

$$\widehat{\nabla}_G J(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{H-1} \gamma^t r_{t+1}^{(i)} \sum_{h=0}^{t} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_h^{(i)} | s_h^{(i)}) \right). \tag{3.77}$$

[7] To avoid measurability issues, take $\mathcal{T}$ to be the set of *realizable* trajectories, i.e., only including initial states from the support of μ and next states from the support of p. We still need the policy to be full-support stochastic, as in the PGT, for (3.75) to be well defined.

[8] This is only true in episodic approaches like REINFORCE, where the policy parameters are fixed for the whole trajectory (or batch). In fully online methods, future actions depend on past rewards since the agent modifies its policy depending on the feedback it receives.

We provide our own proof of the unbiasedness of the G(PO)MDP estimator. First, we write its expected value as:

$$\begin{aligned}
\mathop{\mathbb{E}}_{\mathcal{D}\sim p_{\boldsymbol{\theta}}}\left[\widehat{\nabla}_G J(\boldsymbol{\theta};\mathcal{D})\right] &= \mathop{\mathbb{E}}_{\mathcal{D}\sim p_{\boldsymbol{\theta}}}\left[\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H-1}\gamma^t r_{t+1}^{(i)}\left(\sum_{h=0}^{H-1}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_h^{(i)}|s_h^{(i)})\right.\right.\\
&\qquad\left.\left.-\sum_{h=t+1}^{H-1}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_h^{(i)}|s_h^{(i)})\right)\right]\\
&= \mathop{\mathbb{E}}_{\mathcal{D}\sim p_{\boldsymbol{\theta}}}\left[\widehat{\nabla}_R J(\boldsymbol{\theta};\mathcal{D})\right]\\
&\qquad-\mathop{\mathbb{E}}_{\mathcal{D}\sim p_{\boldsymbol{\theta}}}\left[\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{H-1}\gamma^t r_{t+1}^{(i)}\sum_{h=t+1}^{H-1}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_h^{(i)}|s_h^{(i)})\right)\right]\\
&= \nabla J(\boldsymbol{\theta})-\mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{H-1}\gamma^t r_{t+1}\sum_{h=t+1}^{H-1}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right]. \qquad (3.78)
\end{aligned}$$

Then, we show that the second term in (3.78) is null:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{H-1}\gamma^t r_{t+1}\sum_{h=t+1}^{H-1}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right]\\
&\quad=\sum_{t=0}^{H-1}\gamma^t\mathop{\mathbb{E}}_{\tau_{0:t}\sim p_{\boldsymbol{\theta}}}\left[r_{t+1}\sum_{h=t+1}^{H-1}\mathop{\mathbb{E}}_{\tau_{t+1:H-1}\sim p_{\boldsymbol{\theta}}}\left[\mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right]\right]\right]\\
&\quad=0, \qquad (3.79)
\end{aligned}$$

where the last equality is from (3.7).

The purpose of removing some unnecessary terms from REINFORCE is to reduce the variance of the estimator. See (Peters and Schaal, 2008b; Zhao et al., 2011; Pirotta et al., 2013a) for a discussion. We provide our own upper bounds on the variance of REINFORCE and G(PO)MDP in Section 5.5, after introducing *smoothing policies*. For now, notice that the variance of both estimators is inversely proportional to the batch size:

$$\mathop{\mathbb{V}\mathrm{ar}}_{\mathcal{D}\sim p_{\boldsymbol{\theta}}}(\widehat{\nabla}_R J(\boldsymbol{\theta};\mathcal{D}))=\frac{1}{N}\mathop{\mathbb{V}\mathrm{ar}}_{\tau\sim p_{\boldsymbol{\theta}}}(\widehat{\nabla}_R J(\boldsymbol{\theta};\{\tau\})), \qquad (3.80)$$

since the trajectories of the batch are i.i.d.. The same holds for $\widehat{\nabla}_G J$ and any other estimator that is computed as a sample average. Using a large batch size is a way to reduce the variance of the gradient estimator. The resulting policy updates are more stable, but also less frequent.[9] We analyze this trade-off in Section 5.6.

[9] In simulation, we can generate several trajectories in parallel. This is hardly possible in real-world applications.

A remark on partial observability

3.5.3 Equivalence with PGT

Consider a minibatch, finite-horizon version of the gradient estimator used in Monte Carlo PGT (3.64):

$$\begin{aligned}\widehat{\nabla}_P J(\boldsymbol{\theta};\mathcal{D}) &= \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{H-1}\gamma^t\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_t^{(i)}|s_t^{(i)})\sum_{h=t}^{H-1}\gamma^{h-t}r_{h+1}\right)\\ &= \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{H-1}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_t^{(i)}|s_t^{(i)})\sum_{h=t}^{H-1}\gamma^{h}r_{h+1}\right).\end{aligned} \tag{3.81}$$

As observed by Peters and Schaal (2008b), this is numerically equivalent to the G(PO)MDP estimator (3.77). The equivalence is from reordering the terms in the nested summations of (3.81) (see Lemma B.1). We will adopt the G(PO)MDP form in the following.

3.5.4 Baselines for variance reduction

The concept of baseline introduced in Equation 3.56 can be used as a variance-reduction technique. For any constant (possibly vector-valued) baseline b, the modified REINFORCE estimator:

$$\widehat{\nabla}_R J(\boldsymbol{\theta};\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{H-1}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=0}^{H-1}\gamma^t r_{t+1}^{(i)} - b\right), \tag{3.82}$$

is still unbiased since:

$$\mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\left(\sum_{t=0}^{H-1}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\right)b\right] = \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(\tau)\right]b = 0. \tag{3.83}$$

A simple, but effective choice for b is the expected return:

$$b^{AVG} = \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[R(\tau)\right], \tag{3.84}$$

which has the effect of centering the returns around zero. Intuitively, this should remove the variance due to outliers. Peters and Schaal (2008b) obtain a variance-minimizing vector-valued baseline for reinforce by explicitly solving the following

problem for $i = 1, \dots, d$ (where $\boldsymbol{\theta} \in \mathbb{R}^d$):

$$\begin{aligned} b^R[i] &= \arg\min_b \mathop{\mathbb{V}\mathrm{ar}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\left(\sum_{t=0}^{H-1} \nabla_{\theta_i} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right) (R(\tau) - b) \right] \\ &= \arg\min_b \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\left(\sum_{t=0}^{H-1} \nabla_{\theta_i} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right)^2 (R(\tau) - b)^2 \right] \qquad (3.85) \\ &= \frac{\mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\left(\sum_{t=0}^{H-1} \nabla_{\theta_i} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right)^2 R(\tau) \right]}{\mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\left(\sum_{t=0}^{H-1} \nabla_{\theta_i} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right)^2 \right]}, \qquad (3.86) \end{aligned}$$

where (3.85) is because the first moment is independent from the baseline. Note how b_R is just a weighted version of (3.84). By constraining the baseline to be a scalar, we obtain the following:

$$b^R = \frac{\mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\left\| \sum_{t=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right\|^2 R(\tau) \right]}{\mathbb{E}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\left\| \sum_{t=0}^{H-1} \nabla_{\theta_i} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right\|^2 \right]}, \qquad (3.87)$$

which has less variance-reduction power than (3.86), but can ease the computation of (3.82).

For G(PO)MDP, making the baseline time-dependent can be advantageous. For any $(b_t)_{t=0}^{H-1}$ such that b_t is independent from $\tau_{0:t}$, the following:

$$\widehat{\nabla}_G J(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{H-1} \left(\gamma^t r_{t+1}^{(i)} - b_t \right) \sum_{h=0}^{t} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_h^{(i)}|s_h^{(i)}) \right), \qquad (3.88)$$

is unbiased since:

$$\begin{aligned} &\mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\left(\sum_{t=0}^{H-1} b_t \sum_{h=0}^{t} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_h|s_h) \right) \right] \\ &\quad = \sum_{t=0}^{H-1} b_t \mathop{\mathbb{E}}_{\tau_{0:t} \sim p_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau_{0:t}) \right] = 0. \qquad (3.89) \end{aligned}$$

The time-dependent equivalent of the average-return baseline (3.84) is the average-reward one:

$$b_t^{AVG} = \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\gamma^t r_{t+1} \right]. \qquad (3.90)$$

Peters and Schaal (2008b) suggest to use a vector-valued baseline that minimizes

the variance *step-wise*:

$$b_t^G[i] = \arg\min_b \mathbb{V}\underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathrm{ar}}\left[\left(\gamma^t r_{t+1} - b\right)\sum_{h=0}^{t}\nabla_{\theta_i}\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right]$$
$$= \frac{\mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\left(\sum_{h=0}^{t}\nabla_{\theta_i}\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right)^2\gamma^t r_{t+1}\right]}{\mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\left(\sum_{h=0}^{t}\nabla_{\theta_i}\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right)^2\right]}, \tag{3.91}$$

which is a weighted version of the average-reward baseline (3.90). This cannot be considered a variance-minimizing baseline in the sense of (3.86), since it neglects possible inter-time correlations, but is effective in practice (Peters and Schaal, 2008b). All the baselines we have proposed cannot be computed exactly and must be estimated from data. To keep the gradient estimate unbiased, separate data should be used for the baseline. In the case of variance-minimizing baselines, separate data should also be used for the numerator and the denominator. In practice, a single batch of data is commonly used, trusting that the bias will have a negligible effect on the estimation error, compared to the variance reduction granted by a larger batch.

3.5.5 Off-policy policy gradient

The actor-only PG algorithms presented in this section can be adapted to the off-policy setting by means of importance sampling. Let $\mathcal{D}\sim p_{\pi_B}$ be a set of N trajectories collected with behavioral policy π_B, and $\pi_{\boldsymbol{\theta}} \ll \pi_B$ be our target policy. Then from (3.75):

$$\nabla J(\boldsymbol{\theta}) = \underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathbb{E}}\left[\nabla\log p_{\boldsymbol{\theta}}(\tau)R(\tau)\right] = \underset{\tau\sim p_{\pi_B}}{\mathbb{E}}\left[w_{\pi_{\boldsymbol{\theta}}/\pi_B}(\tau)\nabla\log p_{\boldsymbol{\theta}}(\tau)R(\tau)\right], \tag{3.92}$$

where $w_{\pi_{\boldsymbol{\theta}}/\pi_B}(\tau)$ is an importance weight defined as in (2.43). From this, we obtain the off-policy version of the REINFORCE estimator (Mastrangelo, 2015):

$$\widehat{\nabla}_R J(\boldsymbol{\theta};\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\underbrace{\left(\prod_{t=0}^{H-1}\frac{\pi_B(a_t^{(i)}|s_t^{(i)})}{\pi_{\boldsymbol{\theta}}(a_t^{(i)}|s_t^{(i)})}\right)}_{w_{\pi_{\boldsymbol{\theta}}/\pi_B}}\left(\sum_{t=0}^{H-1}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=0}^{H-1}\gamma^t r_{t+1}^{(i)}\right), \tag{3.93}$$

which is still unbiased but can have much larger variance than its on-policy counterpart. This is worse for a longer horizon H, due to the long product in the definition of the importance weight. Interestingly, the off-policy version of G(PO)MDP allows to compute importance weights on partial trajectories (see Mastrangelo, 2015, for a derivation):

$$\widehat{\nabla}_G J(\boldsymbol{\theta};\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{H-1}\gamma^t r_{t+1}^{(i)}\underbrace{\left(\prod_{h=0}^{t}\frac{\pi_B(a_h^{(i)}|s_h^{(i)})}{\pi_{\boldsymbol{\theta}}(a_h^{(i)}|s_h^{(i)})}\right)}_{\widetilde{w}_{\pi_{\boldsymbol{\theta}}/\pi_b}}\sum_{h=0}^{t}\nabla_{\boldsymbol{\theta}}\log\pi_{\boldsymbol{\theta}}(a_h^{(i)}|s_h^{(i)})\right), \tag{3.94}$$

where $\widetilde{w}_{\pi_{\boldsymbol{\theta}}/\pi_B}$ is known as per-decision importance weight (Precup et al., 2000). Mastrangelo (2015) also derives variance minimizing baselines for off-policy REINFORCE and G(PO)MDP, which are just importance-weighted versions of (3.86) and (3.91). Still, the variance injected by importance weights can have disastrous effects on the learning process, especially if π_B and $\pi_{\boldsymbol{\theta}}$ are quite different. In Chapter 6, we propose a more principled approach to off-policy policy optimization that takes this *distributional mismatch* issue specifically into account.

An off-policy *actor-critic* (Section 3.5) algorithm was proposed by Degris et al. (2012) that uses importance sampling for the actor and gradient-TD for the critic.

3.6 Natural Policy Gradient

The policy gradient $J(\boldsymbol{\theta})$ depends on the specific policy parametrization. This can lead to inefficient policy updates, where changing the parameters (at the price of expensive sample experience) has a negligible effect on the policy or, conversely, a small parameter update can yield a radically different policy with unexpected and possibly unsafe behavior. The *natural gradient* approach (Amari, 1998) aims to alleviate this problem. First, consider the abstract problem of maximizing a (non-convex) functional $f : \Delta_{\mathcal{X}} \to \mathbb{R}$ over a set of parametric distributions $\{q_{\boldsymbol{\theta}} \in \Delta_{\mathcal{X}} | \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^d\}$ over measurable space $\mathcal{X}$. Amari (1998) proposes to follow the direction of maximum improvement in the space of distributions $\Delta_{\mathcal{X}}$ rather than in parameter space Θ. Since we can only act on parameters after all, this is done indirectly by a *change of metric*:

$$\widetilde{\nabla}_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = F(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}), \tag{3.95}$$

where $F(\boldsymbol{\theta})$ is the FIM of $q_{\boldsymbol{\theta}}$ and $\widetilde{\nabla}_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$ is called *natural gradient*. This definition is better understood in the framework of information geometry (Amari, 2016). However, recall (Section 3.3.1) that the FIM is invariant under sufficient statistics and can be used to measure distances between distributions. The natural gradient (3.95) is indeed the steepest ascent direction in the Riemannian space of parametric distributions (Amari, 1998, Theorem 1).

The natural gradient approach was applied to the policy optimization problem by Kakade (2001a), yielding *Natural Policy Gradient* (NPG). Monte Carlo PGT (Algorithm 1) and actor-critic algorithms (Section 3.7) can be modified by replacing *vanilla*[10] policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ with:

$$\widetilde{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = F(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{3.96}$$

where $F(\boldsymbol{\theta})$, defined in Equation 3.10, is the FIM of the distribution of indefinite trajectories (Bagnell and Schneider, 2003, see also our Equation 3.14). Natural policy gradient updates are indeed covariant w.r.t. policy parametrization (Peters and Schaal, 2008a, Theorem 1).

[10] Plain policy gradient ∇J is often called *vanilla* in comparison to NPG *as it keeps the 'vanilla flavor' of the policy* [sic] (Peters, 2010).

Actor-only policy gradient (Algorithm 2) can be modified in the same way, but $F(\boldsymbol{\theta})$ is now the FIM of the distribution $p_{\boldsymbol{\theta}}$ of finite trajectories (2.30). Using (3.10) and the log trick, it can be written simply as:

$$F(\boldsymbol{\theta}) = - \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{H-1} \nabla^2_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right]. \tag{3.97}$$

Of course, the FIM needs to be estimated as well, and (3.97) provides a rather convenient form. However, for high-dimensional policy parameters (e.g., for deep neural policies), explicit Fisher estimation may be computationally unfeasible or affected by very large variance. The *conjugate gradient method* can be used to estimate the natural policy gradient directly, leading to the *Truncated Natural Policy Gradient* (TNPG) algorithm (Duan et al., 2016; Schulman et al., 2015a).

3.7 Actor-Critic Algorithms

Value function estimation can be used to improve policy gradient estimation. A value function approximator used in such an ancillary way is called a *critic* (Barto et al., 1983). Algorithms based on this twofold learning strategy are called *Actor Critic* (AC) (Konda and Tsitsiklis, 2000), where the *actor* is the policy that is being optimized. As already stressed, AC is *not* value-based learning since value function approximation is subordinate to policy gradient estimation.

The purpose of using a critic is to reduce the variance of policy gradient estimates. In fact, the simpler use of a critic is as a variance-reducing baseline in Monte Carlo PGT. From Equation 3.57 and (3.56), we can replace $V^{\boldsymbol{\theta}}$ with a critic $V^{\boldsymbol{\omega}} : \mathcal{S} \to \mathbb{R}$ without introducing any bias, as long as the latter does not depend on actions. Carefully designed action-dependent baselines (Gu et al., 2017; Grathwohl et al., 2018; Liu et al., 2018a; Wu et al., 2018) are also possible, although their usefulness is controversial (Tucker et al., 2018). The critic parameters $\boldsymbol{\omega}$ can be learned with any policy evaluation technique such as the ones described in Section 2.4.1.

More typically, the Monte Carlo estimate of the quality function $Q^{\boldsymbol{\theta}}$ is replaced with a critic $Q^{\boldsymbol{\omega}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, often at the cost of introducing some bias in the policy gradient estimate. This allows to update policy parameters in a fully online manner. We outline the prototypical actor-critic policy gradient method in Algorithm 3, where b_t is a baseline. In turn, this baseline can be replaced by a value-function critic V^{ξ} with separate parameters. Alternatively, from Equation 3.58, we can use a single critic $A^{\boldsymbol{\omega}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for the advantage function.

3.7.1 Compatible actor-critic

A special case of actor-critic PG, which maintains the unbiasedness of the policy gradient estimate, was proposed by Sutton et al. (1999). It requires to use a so-called *compatible critic*:[11]

$$Q^{\boldsymbol{\omega}}(s,a) = \boldsymbol{\omega}^T \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s). \tag{3.98}$$

[11] Actually, the compatible critic can have an additional constant (w.r.t. $\boldsymbol{\omega}$) additive term. We prefer to subsume it in the baseline.

Algorithm 3 Actor-Critic Policy Gradient

1: **Input:** initial policy parameters $\boldsymbol{\theta}_0$, initial critic parameters $\boldsymbol{\omega}_0$, step size sequence $(\alpha_t)_{t=0}^{\infty}$
2: Observe initial state s_0
3: **for** $t = 0, 1, 2 \ldots$ **do**
4: Perform action $a_t \sim \pi_{\boldsymbol{\theta}_t}(\cdot|s_t)$
5: Observe reward r_{t+1} and next state s_{t+1}
6: $\widehat{\nabla} J(\boldsymbol{\theta}_t) = \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}_t}(a_t|s_t)(Q^{\boldsymbol{\omega}_t}(s_t, a_t) - b_t)$
7: $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha_t \widehat{\nabla} J(\boldsymbol{\theta}_t)$
8: Update critic parameters $\boldsymbol{\omega}_t$ into $\boldsymbol{\omega}_{t+1}$
9: **end for**

Provided $\boldsymbol{\omega}_t$ is a local optimum of the following MSE (the Q-function analog of (2.32)):

$$L(\boldsymbol{\omega}; \boldsymbol{\theta}) = \mathop{\mathbb{E}}_{s,a \sim \nu_\mu^{\boldsymbol{\theta}}} \left[\left(Q^{\boldsymbol{\theta}}(s,a) - Q^{\boldsymbol{\omega}}(s,a) \right)^2 \right], \tag{3.99}$$

that is, $\nabla_{\boldsymbol{\omega}} L(\boldsymbol{\omega}_t; \boldsymbol{\theta}_t) = 0$, the policy gradient estimate from Line 6 of Algorithm 3 is unbiased (Sutton et al., 1999, Theorem 2), in the sense that:

$$\mathop{\mathbb{E}}_{s,a \sim \nu_\mu^{\boldsymbol{\theta}}} \left[\widehat{\nabla} J(\boldsymbol{\theta}_t) \right] = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t). \tag{3.100}$$

An immediate corollary is that actor-critic PG with a compatible, locally-optimal critic inherits the convergence properties of Monte Carlo PGT.

3.7.2 Convergence of actor-critic algorithms

The convergence of general actor critic algorithms is still largely an open problem. Even compatible actor-critic does not specify the way in which a local minimizer of the critic error (3.99) can be found. Typically, this involves an iterative procedure such as SGD:

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t - \alpha_C \left(Q^{\boldsymbol{\omega}}(s_t, a_t) - Q^{\boldsymbol{\theta}}(s_t, a_t) \right) \nabla_{\boldsymbol{\omega}} Q^{\boldsymbol{\omega}_t}(s_t, a_t), \tag{3.101}$$

where α_C is a learning rate for the critic. To guarantee $\nabla_{\boldsymbol{\omega}} L(\boldsymbol{\omega}_t) = 0$, we need to iterate (3.101) for many steps, which requires to delay the update of policy parameters or, equivalently, to collect much more data than what appears from the synthetic pseudocode of Algorithm 3. This approach is sometimes called *decoupled AC* (Wu et al., 2020) and is mainly of theoretical interest. In practice, it is more common to update the policy and critic parameters simultaneously. To simulate the fact that one policy update should correspond to several critic updates, a larger learning rate is used for updating critic parameters, i.e., $\alpha_C \gg \alpha$. This is known as *two-timescale* AC. Classic convergence proofs for two-timescale AC are based on ordinary differential equations (Borkar and Konda, 1997; Konda and Tsitsiklis, 2000) and only guarantee asymptotic convergence. In very recent work, Wu et al.

(2020) also provide a finite sample analysis of two-timescale AC. Unfortunately, the convergence rates they provide are worse than those of REINFORCE (cf. Chapter 3). At the time of writing, further work is needed to fully justify AC approaches from a theoretical perspective. However, the practical advantages of AC algorithms are widely recognized. We show some examples in Section 3.7.4, 3.9 and 3.10.

3.7.3 Generalized advantage estimation

The update of critic parameters (Algorithm 3, Line 6) is purposefully left vague as it may be performed with different policy evaluation techniques. We report here an approach developed by Schulman et al. (2015b) called *Generalized Advantage Estimation* (GAE) and inspired by eligibility traces (Sutton and Barto, 2018). It allows to form an advantage function estimate from a value-function critic $V^{\boldsymbol{\omega}}$ obtained, for instance, with MC or TD evaluation (Section 2.7).[12] The GAE estimator is just a discounted sum of TD errors:

$$A^{\boldsymbol{\omega}}_{\lambda}(s_t, a_t) = \sum_{h=0}^{\infty} (\gamma\lambda)^h \delta^{\boldsymbol{\omega}}_{t+h}(s_t), \tag{3.102}$$

where $\lambda \in [0, 1]$ is a bias-variance trade-off parameter. The case $\lambda = 1$ corresponds to MC evaluation:

$$A^{\boldsymbol{\omega}}_{1}(s_t, a_t) = G_t - V^{\boldsymbol{\omega}}(s_t), \tag{3.103}$$

hence is unbiased but suffers from the highest variance. The other extreme, $\lambda = 0$, yields $A^{\boldsymbol{\omega}}_{0}(s_t, a_t) = \delta^{\boldsymbol{\omega}}_{t}(s_t)$, which is only unbiased when the V-function estimate is, since:

$$\mathbb{E}\left[\delta^{\boldsymbol{\omega}}_{t}(s_t)\right] = \mathbb{E}\left[r_t + \gamma V^{\boldsymbol{\omega}}(s_{t+1}) - V^{\boldsymbol{\omega}}(s_t)\right] = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t), \tag{3.104}$$

when $\mathbb{E}\left[V^{\boldsymbol{\omega}}\right] = V^{\pi}$. It is otherwise biased, but it removes most of the variance of G_t. In general, $\lambda < 1$ can be used to trade-off bias and variance.

3.7.4 Natural Actor-Critic

Kakade (2001a) pointed out a surprising relationship between natural policy gradient and the compatible critic. Let $Q^{\boldsymbol{\omega}} : s, a \mapsto \boldsymbol{\omega}^T \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)$ be the compatible critic for policy $\pi_{\boldsymbol{\theta}}$. Let $\widetilde{\boldsymbol{\omega}}$ be a (local) minimizer of the critic MSE (3.99), i.e., $\nabla_{\boldsymbol{\omega}} L(\boldsymbol{\omega}; \boldsymbol{\theta}) = 0$. Then (Kakade, 2001a, Theorem 1):

$$\widetilde{\boldsymbol{\omega}} = \widetilde{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{3.105}$$

i.e., the natural policy gradient (assumed to be well defined) *is* the optimal compatible critic parameter.

This fact was exploited by Peters and Schaal (2008a) for their Natural Actor Critic algorithm. They use a variant of LSTD(λ) (Boyan, 2002) to learn the

[12] Note that the advantage function *cannot* be learned with TD evaluation directly, as observed, e.g., by Peters and Schaal (2008a)

parameters $\boldsymbol{\omega}$ of the compatible critic and those of a parametric value-function baseline $V^{\boldsymbol{v}} : s \mapsto \boldsymbol{v}^T \phi(s)$ simultaneously. Q-function parameters are directly used to update policy parameters $\boldsymbol{\theta}$ in the natural gradient direction, using (3.105). The episodic variant of this algorithm, replacing the V-function critic with a MC value estimate, is called eNAC (episodic Natural Actor Critic) and was used by Peters and Schaal (2008a) to learn motor skills for robot baseball.

3.8 Trust-Region Methods

The policy gradient direction computed (or estimated) under policy $\pi_{\boldsymbol{\theta}}$, due to its local (first-order) nature, does not take into account the fact that the updated policy parameters $\boldsymbol{\theta'} = \boldsymbol{\theta} + \alpha \nabla J(\boldsymbol{\theta})$ will induce a possibly very different distribution $\nu_\mu^{\boldsymbol{\theta'}}$ over states and actions. This *distributional mismatch* issue, peculiar of RL, can be a source of performance oscillations and unexpected, potentially unsafe behavior (Kakade and Langford, 2002). Trust-region methods constrain the updated parameter vector to lie within a sufficiently small neighborhood of the previous one, according to a meaningful metric.

3.8.1 TRPO

The *Trust Region Policy Optimization* (TRPO) algorithm by Schulman et al. (2015a) approximately solves the following constrained optimization problem:

$$\begin{aligned} &\max_{\boldsymbol{\theta'}} J(\boldsymbol{\theta'}) \\ &\text{s.t. } \mathcal{KL}(\nu_\mu^{\boldsymbol{\theta}} \| \nu_\mu^{\boldsymbol{\theta'}}) \leqslant \delta, \end{aligned} \tag{3.106}$$

where $\boldsymbol{\theta}$ are the current policy parameters and $\delta > 0$ is a small threshold. The constraint defines a *trust region* in the space of candidate policies, ensuring that their induced state-action occupancies lie in the *proximity* of the data distribution. From the performance difference Lemma (3.4), using importance sampling, the objective can be rewritten as:

$$\max_{\boldsymbol{\theta'}} J(\boldsymbol{\theta'}) - J(\boldsymbol{\theta}) = \max_{\boldsymbol{\theta'}} \mathop{\mathbb{E}}_{s \sim \nu_\mu^{\boldsymbol{\theta'}}} \left[A^{\boldsymbol{\theta}}(s,a) \right] = \mathop{\mathbb{E}}_{\substack{s \sim d_\mu^{\boldsymbol{\theta'}} \\ a \sim \pi_{\boldsymbol{\theta}}}} \left[\frac{\pi_{\boldsymbol{\theta'}}(a|s)}{\pi_{\boldsymbol{\theta}}(a|s)} A^{\boldsymbol{\theta}}(s,a) \right]. \tag{3.107}$$

Then, some approximations are introduced: the state distribution under $\pi_{\boldsymbol{\theta'}}$ is replaced with the one under $\pi_{\boldsymbol{\theta}}$ in virtue of the constraint. A second order approximation (cf. Equation 3.11) is applied to the constraint itself, obtaining:

$$\begin{aligned} &\max_{\boldsymbol{\theta'}} g(\boldsymbol{\theta'}) \coloneqq \mathop{\mathbb{E}}_{s,a \sim \nu_\mu^{\boldsymbol{\theta}}} \left[w_{\boldsymbol{\theta'}/\boldsymbol{\theta}}(s,a) A^{\boldsymbol{\theta}}(s,a) \right] \\ &\text{s.t. } \left\| \boldsymbol{\theta'} - \boldsymbol{\theta} \right\|_{F(\boldsymbol{\theta})}^2 \leqslant 2\delta, \end{aligned} \tag{3.108}$$

where $w_{\boldsymbol{\theta'}/\boldsymbol{\theta}}(s,a) = \frac{\pi_{\boldsymbol{\theta'}}(a|s)}{\pi_{\boldsymbol{\theta}}(a|s)}$. A *steepest ascent* update is obtained by explicitly solving the constrained problem after a first order expansion of the surrogate

objective $g(\boldsymbol{\theta}')$ at $\boldsymbol{\theta}$ (cf. Peters and Schaal, 2008b):

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha F(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}), \tag{3.109}$$

which satisfies the constraint for sufficiently small step size α. This is just a natural gradient update. In practice, Schulman et al. (2015a) approximately compute the ascent direction via conjugate gradient and select the step size with a line search procedure, after estimating the advantages in (3.108) in a MC fashion. In a follow-up work, Schulman et al. (2015b) employ GAE (Section 3.7.3) to estimate the advantage function.

TRPO was successfully applied to simulated robot locomotion in combination with deep policies (Schulman et al., 2015a) and is considered a state-of-the art deep RL algorithm for continuous control. However, its theoretical foundations are controversial. The original justification of Schulman et al. (2015a) is a *monotonic performance improvement* guarantee, that we discuss in more depth in Chapter 5. However, the several approximations involved in TRPO lead to question its validity as a monotonically improving algorithm.[13] A later stream of literature (Neu et al., 2017; Geist et al., 2019; Shani et al., 2020) interprets TRPO as an approximate mirror descent (Beck and Teboulle, 2003) algorithm, where the KL divergence is the Bregman distance associated to negative entropy. This perspective gives up monotonic improvement guarantees, but explains the empirical success of TRPO in a more natural way, which is also closer to how we have introduced the algorithm in this section. Using the mirror descent formulation, Shani et al. (2020) proved the convergence of TRPO to the *global* optimum in the tabular case (see Section 7.7).

3.8.2 PPO

A further approximation to TRPO yields the *Proximal Policy Optimization* (PPO) algorithm (Schulman et al., 2017b), that is behind some of the most impressive empirical results of policy optimization (Section 3.10). In PPO, the proximity constraint from (3.108) is entirely removed. In turn, the surrogate objective is modified to clip importance weights as follows:

$$g(\boldsymbol{\theta}') = \mathop{\mathbb{E}}_{s,a\sim\nu_{\mu}^{\boldsymbol{\theta}}} \left[\min \left\{ w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(s,a) A^{\boldsymbol{\theta}}(s,a), \widetilde{w}_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(s,a) A^{\boldsymbol{\theta}}(s,a) \right\} \right], \tag{3.110}$$

where $\widetilde{w}_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(s,a) = \min\{\max\{w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(s,a), 1-\epsilon\}, 1+\epsilon\}$ and ϵ is a small constant (a typical default value is $\epsilon = 0.2$). Similarly to the TRPO constraint, the weight clipping has the purpose of penalizing large deviations from the current policy $\pi_{\boldsymbol{\theta}}$, implicitly defining a trust region.[14] The clipped objective is never allowed to be larger than the original one, to prevent overestimation. Besides performing better than TRPO on several benchmark tasks (Schulman et al., 2017b), PPO is computationally less expensive since it does not require to compute a natural gradient direction.

[13] In recent work, Pajarinen et al. (2019) manage to remove some of the approximations of the original TRPO. In particular, the step size for the trust-region update can be computed exactly for exponential-family policies with natural parametrization paired with a compatible critic.

[14] We could also see TRPO as a variant of PPO that implicitly constrains the importance weights (see Section 6.2.4).

3.9 Deterministic Policy Gradients

The policy optimization methods presented so far only apply to stochastic policies. Silver et al. (2014) derived an analog of the Policy Gradient Theorem for deterministic (differentiable) policies of the form $\pi_{\boldsymbol{\theta}} : \mathcal{S} \to \mathcal{A}$. Provided p, $\nabla_a p$, $\nabla_{\boldsymbol{\theta}} \pi$, r, $\nabla_a r$ and μ exist and are continuous w.r.t. states, actions and policy parameters, the *Deterministic Policy Gradient* (DPG) can be expressed as (Silver et al., 2014, Theorem 1):

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{s \sim d_\mu^{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s) \nabla_a Q^{\boldsymbol{\theta}}(s,a)|_{a=\pi_{\boldsymbol{\theta}}(s)} \right], \tag{3.111}$$

which allows to estimate it as an average over visited states. No averaging over actions is needed, which can significantly reduce the variance w.r.t. traditional policy gradient. This is inherently an actor-critic approach, since we need to evaluate the *derivative* of the Q-function w.r.t. actions in sampled state-action pairs. Similarly to the stochastic case, a compatible critic can be defined that preserves the unbiasedness of the gradient estimate (Silver et al., 2014, Theorem 3). Unfortunately, collecting data with the deterministic policy that is being optimized does not provide a sufficient amount of exploration. For this reason, a stochastic behavioral policy π_B is used in its place. This amounts to estimating the following quantity:

$$\mathop{\mathbb{E}}_{s \sim d_\mu^{\pi_B}} \left[\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(s) \nabla_a Q^{\boldsymbol{\theta}}(s,a)|_{a=\pi_{\boldsymbol{\theta}}(s)} \right] \simeq \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \tag{3.112}$$

which is a deterministic-policy variant of the off-policy policy gradient by Degris et al. (2012). The approximation w.r.t. (3.111), due to a change of state-occupancy measure, is justified by the fact that the zeros of (3.112) are still local optima of $J(\boldsymbol{\theta})$ provided the Q-function approximation is exact (Degris et al., 2012, Theorem 2). In off-policy The behavioral policy used in off-policy DPG is typically a perturbed version of the deterministic policy being optimized, making the approximation even less serious.

Lillicrap et al. (2016) combined off-policy DPG with the stabilization tricks of DQN (see Section 2.8.2), obtaining the *Deep Deterministic Policy Gradient* (DDPG) algorithm. Deep neural networks are used to represent both the deterministic policy and the Q-function critic. The latter is learned via semi-gradient, off-policy temporal difference evaluation.[15] Target networks are used to stabilize the updates like in DQN, except soft updates are used in place of periodical hard copies to slowly update the target weights. To ensure a sufficient amount of exploration, the behavioral policy is obtained by adding Gaussian or Ornstein-Uhlenbeck noise to the deterministic one. The latter is an auto-correlated noise that is believed to facilitate the control of physical systems. Transitions collected by the behavioral policy are stored in a replay buffer, from which a mini-batch is periodically sampled for updating both the actor and the critic. A popular regularization technique from the deep supervised learning literature, called *batch normalization* (Ioffe and Szegedy,

[15] DDPG is often presented as "DQN for continuous actions". We believe this is misleading, since no greedy policy is actually involved.

2015) is also applied to both actor and critic networks. DDPG was successfully applied to simulated robot locomotion (Lillicrap et al., 2016) and is considered one of the main state-of-the art deep RL algorithms together with TRPO and PPO. However, it is known for its extreme sensitivity to hyperparameters, and requires extensive hyper-parameter tuning to be used for new tasks.

Fujimoto et al. (2018) observed a problem of overestimation bias in DDPG, and addressed it by adapting the Double Q-learning technique (van Hasselt, 2010) to the actor-critic framework. Their algorithm, called TD3 (Twin Delayed Deep Deterministic policy gradient), was shown to outperform DDPG on several benchmark tasks. Further tricks were introduced, such as updating the policy network less often than the critic network to improve stability and perturbing the deterministic action with zero-mean noise in temporal-difference targets to smooth out the critic and reduce policy overfitting.

Ciosek and Whiteson (2020) provide a general policy gradient theorem that generalizes both Sutton's PGT and its deterministic counterpart:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{s\sim d_{\mu}^{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} V^{\boldsymbol{\theta}}(s) - \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} Q^{\boldsymbol{\theta}}(s,a)\, \mathrm{d}a \right], \tag{3.113}$$

for any (deterministic *or* stochastic) policy, provided the V-function is bounded, continuously differentiable in the policy parameters and measurable in the state. The key intuition of Ciosek and Whiteson (2020), inspired by Expected SARSA (John, 1994; van Seijen et al., 2009), is to explicitly compute the argument of the expectation in (3.113), without resorting to statistical estimation. This allows to apply the variance-reduction property of DPG to stochastic policies as well. Like DPG, this is inherently an actor-critic approach. The integral over actions can be exactly computed for some combinations of policy and critic (later extended by Fellows et al., 2018, using Fourier analysis), or approximated by numerical quadrature.

3.10 Policy Gradient at Scale

The versatility of PG algorithms make them applicable to a wide range of control problems, including ones with very high-dimensional state spaces. Algorithms like A3C (Asynchronous Advantage Actor Critic, Mnih et al., 2016) are specifically designed for training large models (policies and critics represented by deep neural networks), with computational *and* sample efficiency in mind. This is at the expense of losing theoretical guarantees, although many of these methods are loosely inspired by theoretically-sound solutions.

A3C is an actor-critic algorithm where the policy and V-function networks share the first layers of weights. The latter is a common trick in deep RL, motivated by the *inductive bias* that actor and critic will likely need the same low-level features. A V-function critic, learned with n-step semi-gradient TD, is used to estimate the advantage function as follows:

$$\widehat{A}^{\boldsymbol{\theta}}(s_t, a_t) = \sum_{h=0}^{n-1} \gamma^h r_{t+h} + \gamma^n V^{\boldsymbol{\omega}}(s_{t+n}) - V^{\boldsymbol{\omega}}(s_t), \tag{3.114}$$

where $n < H$ is a hyperparameter, and represents another way to trade-off bias and variance in advantage estimation similarly to GAE ($n = 1$ corresponds to TD, $n = \infty$ to MC). In turn, the approximate advantage function is used to estimate policy gradients (cf. Equation 3.58). What makes A3C really special is that it employs multiple parallel learners[16] that collect transitions independently and asynchronously apply their policy gradient updates to a central actor-critic model. Besides speeding up computations, this multi-thread approach also improves exploration by diversifying the collected experience. Clearly, this approach heavily relies on simulation and has no practical equivalent in real-world environments. A3C achieved (in some cases surpassed) state-of-the-art performance while training for less time on both Atari games and robot locomotion tasks. It also solved a challenging 3D labyrinth navigation task based on visual input (Mnih et al., 2016).

Researchers later discovered that a synchronous variant of A3C, implemented with a single learner, actually performs better. Moreover, computational efficiency can also be improved by using a GPU. This algorithm is simply called A2C (Advantage Actor Critic) and was mentioned for instance by Wang et al. (2016a).

ACKTR (Actor-Critic using Kronecker-factored Trust Region, Wu et al., 2017) is a trust-region variant of A2C that employs K-FAC (Kronecker-Factored Approximated Curvature, Martens and Grosse, 2015) to compute the natural gradient in a scalable way. It can also be seen as a variant of TRPO where the advantage is estimated as in (3.114) and the trust region computation is made more efficient. A concurrent work by Wang et al. (2017) achieves the same result by linearizing the KL constraint for a suitably designed policy network. Their ACER (Actor Critic with Experience Replay) algorithm also employs a replay buffer, an advanced off-policy RL technique called Retrace (Munos et al., 2016), truncated importance weights with bias correction, and dueling networks (Wang et al., 2016b).

An actor-critic architecture able to scale to *thousands* of machines was proposed by Espeholt et al. (2018). Similarly to A3C, several parallel agents independently interact with a simulated environment. Instead of applying weight updates to a central model, they communicate trajectories to a single central learner running on GPU. Delays in communicating back the updated policy parameters to agents result in off-policy learning, tackled with a novel technique called *V-trace*. The algorithm, called IMPALA (Importance Weighted Actor-Learner Architecture), was successfully applied to the *multi-task reinforcement learning*[17] of a great variety of discrete *and* continuous-action problems.

3.11 Entropy Regularization

As observed by Kakade and Langford (2002), policy gradient methods can struggle to optimize sparse rewards. If the initial policy has a small probability of obtaining a nonzero reward, the gradient will be too small to be of any use. For the same

[16] The implementation suggested by (Mnih et al., 2016) uses multiple CPU threads on the same machine to reduce communications costs.

[17] This means to learn a single policy for several tasks (e.g., all Atari games), which poses a significant scalability challenge.

reason, policy gradient methods can get stuck in bad local optima. These problems can be alleviated by using a stochastic policy that ensures a sufficient amount of exploration (Ahmed et al., 2019). This poses a particular flavor of the ubiquitous exploration-exploitation dilemma: we would like our agent to converge to deterministic behavior in the end since random behavior is always suboptimal in standard MDPs. However, an agent that has control over its own stochasticity will likely commit to a deterministic policy too soon, favoring immediate performance improvement, especially under a first-order optimization strategy, like policy gradient, that relies on local information.

Entropy regularization (Schulman et al., 2017a) addresses these problems by explicitly rewarding stochastic behavior. The reward function is modified as follows:

$$\widetilde{r}_t = (1-\tau) r_t + \tau \mathbb{H}\left(\pi_{\boldsymbol{\theta}}(\cdot|s_t)\right), \tag{3.115}$$

where $\mathbb{H}\left(\pi(\cdot|s_t)\right) = \mathbb{E}_{a\sim\pi(\cdot|s_t)}\left[-\log \pi(a|s_t)\right]$ is the entropy of the action distribution at state s_t and $\tau \in [0, 1)$ is a regularization coefficient. Note how this modified reward signal depends on the current policy $\pi_{\boldsymbol{\theta}}$, hence is non-stationary from the perspective of a learning agent. Actor-only policy gradient algorithms can be easily modified to account for the entropy bonus (e.g., Ahmed et al., 2019). However, entropy regularization finds its most effective application in actor-critic algorithms, where a properly defined value function can guide the agent to uniformly explore the state space (Haarnoja et al., 2017). Haarnoja et al. (2018) propose a *Soft Actor Critic* (SAC) algorithm and show that entropy regularization can significantly improve performance on complex tasks such has controlling a simulated humanoid robot.

From the theoretical perspective, entropy regularization can make the policy optimization problem easier (Ahmed et al., 2019; Neu et al., 2017; Shani et al., 2020; Mei et al., 2020; Agarwal et al., 2020) at the price of introducing some bias to the final solution, due to the modification of the reward function.

In Chapter 8 we propose an alternative approach to adaptive exploration within the framework of safe policy gradients.

3.12 Beyond Policy Gradients

In this Section, we present some approaches to policy optimization that are not based on REINFORCE (Williams, 1992) or the policy gradient theorem (Sutton et al., 1999). These techniques are typically applied to robot control (Deisenroth et al., 2013).

3.12.1 Finite-difference methods

Before the widespread adoption of likelihood-ratio approaches, finite-difference methods were used to estimate policy gradients (e.g., Ng and Jordan, 2000). The idea comes from stochastic simulation literature. To estimate $\nabla J(\boldsymbol{\theta})$, collect N trajectories $\tau_1, \dots \tau_N$, using different perturbed policy parameters $\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_i$ for each τ_i, where $\Delta\boldsymbol{\theta}_i \in \mathbb{R}$ is a small perturbation, and estimate the performance gain

$\Delta J_i = J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_i) - J(\boldsymbol{\theta})$. The policy gradient can be estimated by solving the system of linear equations induced by first order approximations:

$$\Delta\boldsymbol{\theta}_i^T \nabla J(\boldsymbol{\theta}) = \Delta J_i, \tag{3.116}$$

for $i = 1, \ldots, N$. This technique is particularly effective in the simulation of deterministic dynamical systems, and continues to be used in robotics (e.g., Sehnke et al., 2008).

3.12.2 Parameter-based exploration

For episodic tasks, an alternative to REINFORCE-like algorithms is *Policy Gradients with Parameter-based Exploration* (PGPE) (Sehnke et al., 2008). Given a parametric policy class Π_Θ with parameter space $\Theta \in \mathbb{R}^d$, the idea is to define a class of *hyperpolicies* $\mathcal{P}_\mathcal{V} = \{\rho_{\boldsymbol{\nu}} \in \Delta_\Theta | \boldsymbol{\nu} \in \mathcal{V}\}$, that are parametric distributions over policy parameters. The elements of $\mathcal{V} \subseteq \mathbb{R}^m$ are called *hyperparameters* and may have different dimensionality than base policy parameters. The optimization is moved at the level of hyperparameters, and exploration is over policy parameters instead of actions. At the beginning of each episode i, the agent draws policy parameters $\boldsymbol{\theta}_i \sim \rho_{\boldsymbol{\nu}}$, where ν are the current hyperparameters. It then collects a trajectory τ_i with $\pi_{\boldsymbol{\theta}_i}$ and observes return $R(\tau_i)$[18]. A natural measure of performance for hyperpolicies is (with some notational overloading):

$$J(\boldsymbol{\nu}) = \mathop{\mathbb{E}}_{\boldsymbol{\theta}\sim\rho_{\boldsymbol{\nu}}} \left[\mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}} [R(\tau)] \right] = \mathop{\mathbb{E}}_{\boldsymbol{\theta}\sim\rho_{\boldsymbol{\nu}}} [J(\boldsymbol{\theta})] . \tag{3.117}$$

The PGPE algorithm is just REINFORCE on hyperparameters, where the gradient is simply:

$$\nabla_{\boldsymbol{\nu}} J(\boldsymbol{\nu}) = \mathop{\mathbb{E}}_{\substack{\boldsymbol{\theta}\sim\rho_{\boldsymbol{\nu}} \\ \tau\sim p_{\boldsymbol{\theta}}}} [\nabla_{\boldsymbol{\nu}} \log \rho_{\boldsymbol{\nu}}(\boldsymbol{\theta})(R(\tau) - b)] , \tag{3.118}$$

where b is a baseline, and can be estimated in a MC fashion from one or more parameter-trajectory pairs. Pseudo-code for PGPE is provided in Algorithm 4.

Being an instance of stochastic gradient ascent, it has the same convergence guarantees of REINFORCE. However, the gradient estimates have lower variance, which can result in faster convergence. This was the original motivation of Sehnke et al. (2008) and was formally proven by Zhao et al. (2011), who also provide a variance-minimizing baseline. Another advantage is that base policies can be deterministic (since exploration is performed at the level of parameters) and non-differentiable (e.g., Likmeta et al., 2020). PGPE also applies to partially observable MDPs and problems with non-additive rewards, since it only needs to observe the return. This can also result in computational advantages since trajectories need not to be stored. However, PGPE can struggle with very high-dimensional policy parameter spaces, since it does not exploit the special temporal structure of the RL

[18] Assume deterministic rewards for simplicity. The return is then a deterministic function of the trajectory.

Algorithm 4 PGPE (Policy Gradients with Parameter-based Exploration)

Input: initial hyperpolicy parameters $\boldsymbol{\nu}_0$, step size sequence $(\alpha)_{k=0}^{\infty}$, batch size N
for $k = 0, 1 \dots$ **do**
  $\mathcal{D}_k = \varnothing$
  for $i = 1, \dots, N$ **do**
    Sample $\boldsymbol{\theta}_i \sim \rho_{\boldsymbol{\nu}_k}$
    Collect trajectory τ_i with policy $\pi_{\boldsymbol{\theta}_i}$ and observe return R_i
    $\mathcal{D}_k = \mathcal{D}_k \cup \{(\boldsymbol{\theta}_i, R_i)\}$
  end for
  $\widehat{\nabla}_{\boldsymbol{\nu}} J(\boldsymbol{\nu}_k; \mathcal{D}_k) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\nu}} \log \rho_{\boldsymbol{\nu}}(\boldsymbol{\theta}_i)(R_i - b)$
  Update hyperparameters as $\boldsymbol{\nu}_{k+1} \leftarrow \boldsymbol{\nu}_k + \alpha_k \widehat{\nabla}_{\boldsymbol{\nu}} J(\boldsymbol{\nu}_k; \mathcal{D}_k)$
end for

problem. For all these reasons, it is often employed in robotics (Deisenroth et al., 2013).

The hyperpolicy can be any parametric distribution, but Gaussians are by far the most used in practice. A typical setup is to allocate a couple of hyperparameters μ_i, σ_i for each policy parameter θ_i, $i = 1, \dots, d$. Note that $m = 2d$ in this case. The hyperpolicy is then a factored multi-variate Gaussian $\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \mathrm{diag}(\boldsymbol{\sigma}^2))$ and $\boldsymbol{\nu} = [\boldsymbol{\mu} | \boldsymbol{\sigma}]$. Scores can be easily computed, similarly to (3.28). Moreover, the Fisher matrix has a particularly convenient form (Metelli et al., 2020b):

$$F(\boldsymbol{\nu}) = \begin{bmatrix} \mathrm{diag}(\boldsymbol{\sigma})^{-2} & \mathbf{0} \\ \mathbf{0} & 2\mathbb{I} \end{bmatrix}. \tag{3.119}$$

Note that it can be computed exactly and, being diagonal, it is easily inverted.[19] This makes natural gradients particularly enticing in the parameter-based exploration framework (Miyamae et al., 2010).

Parameter-based exploration is also more suitable for off-policy learning (Zhao et al., 2013), at least for moderate-sized policies. The computational advantage is that importance weights are easily obtained as hyperpolicy ratios. The statistical advantage it that, intuitively, the variance of IS estimators scale with the number of policy parameters instead of the task horizon. This idea is further elaborated in Chapter 6.

Since it neglects the internal structure of trajectories, PGPE is often called a *black-box* method. As such, it shares many properties with other black-box algorithms for control such as the *Cross Entropy Method* (CEM) (Rubinstein, 1999; Szita and Lörincz, 2006) and *Evolution Strategies* (ES) (Rechenberg, 1978; Hansen and Ostermeier, 2001), a biologically-inspired approach that was recently found to be competitive with state-of-the-art RL algorithms on modern benchmarks (Salimans et al., 2017).

[19] The Fisher is *block-diagonal* for general multi-variate Gaussian hyperpolicies (Sun et al., 2009), so it admits efficient inversion even with a full covariance matrix (Miyamae et al., 2010).

3.12.3 Policy optimization as inference

The *Expectation Maximization* (EM) approach to policy optimization is to formulate PO as a latent-variable inference problem. In the episodic setting, the latent variable is the trajectory $\tau \sim p_{\boldsymbol{\theta}}$, where $\boldsymbol{\theta}$ are the policy parameters. The observed variable is the return $R(\tau)$ associated to the trajectory. Even in the case of deterministic rewards, a return probability can be artificially defined through a transformation such as $p(R|\tau) \propto R(\tau) - \min_{\widetilde{\tau}} R(\widetilde{\tau})$ or $p(R|\tau) \propto \exp(\beta R(\tau))$ for some $\beta > 0$, so that higher returns are mapped into higher likelihoods. The goal is then to maximize the log-likelihood of the return:

$$\max_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(R) = \log \int p_{\boldsymbol{\theta}}(R, \tau) \, \mathrm{d}\tau. \tag{3.120}$$

The idea of EM (Moon, 1996) is to introduce an auxiliary *variational distribution* $q(\tau)$ that allows to rewrite (3.120) as:

$$\max_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\tau \sim q} \left[\log p_{\boldsymbol{\theta}}(R, \tau) \right] + \mathcal{KL}(q \| p_{\boldsymbol{\theta}}(\tau|R)). \tag{3.121}$$

The algorithm alternates two steps. The expectation step consists in minimizing the KL divergence w.r.t. q. The maximization step consists in maximizing $\mathbb{E}_{\tau \sim q} \left[\log p_{\boldsymbol{\theta}}(R, \tau) \right]$ w.r.t. $\boldsymbol{\theta}$. Convergence to a local maximum of $\log p_{\boldsymbol{\theta}}(R)$ is guaranteed. The Monte Carlo version of (3.121) is just a weighted maximum likelihood estimation problem (Vlassis and Toussaint, 2009):

$$\boldsymbol{\theta}_{k+1} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} p(R|\tau_i) \log p_{\boldsymbol{\theta}}(\tau_i), \tag{3.122}$$

where the trajectories $\tau_1, \ldots, \tau_N$ are independently sampled using the current policy parameters $\boldsymbol{\theta}_k$.

The *Reward Weighted Regression* (RWR) algorithm (Peters and Schaal, 2007; Kober and Peters, 2008) uses a linear policy, which turns (3.122) into a weighted linear regression problem. The PoWER algorithm (Policy Learning by Weighting Exploration with Returns, Kober and Peters, 2008) is a refinement of RWR that employs state-dependent exploration.

An alternative approach consists in explicitly parametrizing the variational distribution. This *variational inference* approach to PO (Neumann, 2011) allows to prevent mode collapse, but is more computationally heavy than Monte Carlo methods.

Overall, the main advantage of EM algorithms with respect to policy gradients is that the former do not require to select a step size.

3.12.4 Relative Entropy Policy Search

The idea of *Relative Entropy Policy Search* (REPS) (Peters et al., 2010; Daniel et al., 2016) is to formulate the policy optimization problem directly in the space of

state-action occupancy distributions::

$$\nu_{k+1} = \arg\max_{\nu} \int_{\mathcal{S}} \int_{\mathcal{A}} \nu(s,a) r(s,a) \, \mathrm{d}s \, \mathrm{d}a$$
$$\text{subject to } \mathcal{KL}(\nu|\nu_k) \leqslant \epsilon, \tag{3.123}$$

with the addition of simplex constraints to ensure ν_{k+1} is a proper probability distribution. The KL constraint ensures that the new solution does not differ too much from the previous one,[20] the same idea of NPG and trust region methods (Sections 3.6, 3.8). This is a convex problem, which has led some to study it in abstract terms (Zimin and Neu, 2013). The next step towards a practical solution is to reformulate (3.123) in terms of policies:

$$\pi_{k+1} = \arg\max_{\pi} \int_{\mathcal{S}} \int_{\mathcal{A}} d^{\pi}(s) \pi(a|s) r(s,a) \, \mathrm{d}s \, \mathrm{d}a$$
$$\text{subject to } \mathcal{KL}(d^{\pi}\pi|\nu_k) \leqslant \epsilon, \tag{3.124}$$

where additional constraints are required to ensure d^{π} is a plausible state-occupancy measure. The problem still admits a closed form solution, which can be obtained with the method of Lagrangian multipliers:

$$\pi_{k+1}(a|s) \propto \nu_k(s,a) \exp\left(\frac{r(s,a) - b}{\eta}\right), \tag{3.125}$$

where b and η depend on Lagrange multipliers and can be found by approximately optimizing the dual function. We omit the precise expression here, but b can intuitively regarded as a baseline and η as a temperature coefficient. Like EM-based approaches, REPS does not require to explicitly set a step size. Moreover, it does not require to restrict the policy space to a parametric class in principle.

In practice, we still need a parametric policy $\pi_{\boldsymbol{\theta}}$ for continuous-action domains. This can be obtained by regressing (3.125) on policy parameters $\boldsymbol{\theta}$, which once again results in a weighted maximum likelihood problem.

3.13 Applications

We conclude our exposition of policy optimization with a brief overview of its applications.

Simulated control benchmarks The most common benchmarks for PO algorithms are simulated continuous control tasks that can be found in the `gym` library[21] by OpenAI (Brockman et al., 2016) or in the `rllab` library (Duan et al., 2016). We employ some of these in our empirical evaluations (see Appendix C for detailed task

[20] The order of the arguments of the KL divergence is important: in this way, the new distribution cannot assign much probability to state-action pairs that were not witnessed by the previous one, which can be seen as a safety measure.

[21] `https://gym.openai.com/`

specifications). The most challenging continuous problems are robot locomotion tasks based on the MuJoCo simulator (Todorov et al., 2012).[22] The realism and actual difficulty of these tasks have been questioned several times (e.g., Recht, 2018).

Videogames Policy gradient algorithms with deep neural policies (e.g., Espeholt et al., 2018) were successfully applied to the Arcade Learning Environment (Bellemare et al., 2013), a suite of over fifty Atari 2600 games which is the common testbed of value-based deep reinforcement learning algorithms (Section 2.8.2). On a much larger scale, PPO was used to beat the world champions of Dota 2, a MOBA (Multiplayer Online Battle Arena) videogame (Berner et al., 2019). In both cases, the agent had only access to the same information as human players, in the form of visual input.

Robotics Policy optimization was also applied to the control of physical robots. The first applications relied on a significant amount of feature engineering (e.g., motor primitives, Peters and Schaal, 2008b). Deisenroth et al. (2013) provides a survey of robotic applications of policy search. A good example is the table-tennis robot by Mülling et al. (2013), which also relies on human demonstrations as a starting point.

More recently, the combination of novel algorithms such as PPO with deep neural policies and increased computational power allowed to train physical robots on raw inputs. Büchler et al. (e.g., 2020) trained a robot arm to play table tennis from scratch. PPO was also used to learn policies for dexterous hand manipulation (Andrychowicz et al., 2020; OpenAI et al., 2019) which can be transfered from a simulated environment to a real robot hand. By also using animal demonstrations as a starting point, Peng et al. (2020a) were able to teach several skills to a quadruped robot.

Other applications Policy gradient algorithms were also applied to autonomous driving (Kendall et al., 2019) and building energy management (Mason and Grijalva, 2019).

[22] Free alternatives are also available (e.g., Ellenberger, 2018–2019).

Safe Reinforcement Learning 4

In this chapter, we review the main approaches to safe reinforcement learning. Safety is a very broad concept, encompassing a wide range of practical concerns. The purpose of this chapter is not to provide an exhaustive survey nor a coherent taxonomy of the several existing approaches, but rather to discuss the main aspects of the safety problem in RL that have been addressed in the literature, to better position our own contributions. Our main focus is always on continuos-control tasks and policy optimization methods, although safe RL approaches are often general enough to be adapted to multiple application domains and algorithmic frameworks. See (García and Fernández, 2015) for a comprehensive survey on safe RL and (Amodei et al., 2016) for a more general one on the safety of artificial intelligence.

The structure of this chapter is simple: after a general introduction on the concept of risk (Section 4.1), we dedicate a section to each of the main aspects of RL safety: agent alignment (Section 4.2), risk-averse RL (Section 4.3), constrained MDPs (Section 4.4), Seldonian RL (Section 4.5), and constrained exploration (Section 4.6). We conclude in Section 4.7 by discussing the perspective adopted in this manuscript, which stems from the literature on monotonic performance improvement (Kakade and Langford, 2002) and combines the concepts of Seldonian RL and safe exploration. Some specific safe RL algorithms that are particularly relevant to our own contributions will be discussed in more depth in the following chapters.

4.1 Three Kinds of Risk

Safety is relevant to reinforcement learning whenever the actions of the agent can have concrete consequences. The purpose of safe RL is to avoid *hazards*, that is, any source of potential damage that may directly or indirectly be caused by the agent during the learning process or after its deployment. The most striking example

is provided by applications of RL to physical systems, such as industrial robots or autonomous cars, which could procure damage to themselves, other artifacts, and, most importantly, humans. However, safety concerns may also arise when RL agents (even software-based ones) are employed to take critical decisions, for instance on treatments patients in healthcare (Thapa et al., 2005; Moore et al., 2014; Zhu et al., 2020) or the management of critical infrastructures (Castelletti et al., 2010).[1]. Undesirable consequences, albeit of a more recoverable nature, are also possible in financial applications in the form o monetary losses (Moody and Saffell, 2001).

A safe RL agent is one that limits *risk*, that is the chance of hazards. This is made hard by the uncertain nature of the environment and of the learning process itself. To better understand the challenge, risk is typically divided into two categories (cf. García and Fernández, 2015):

1. *Inherent risk* is due to the intrinsic stochastic nature of the environment. This includes events that are truly random or too complex to be predicted with certainty, such as pedestrians crossing the road or fluctuations of the stock market.

2. *Epistemic* or *model risk* is due to the incomplete knowledge the agent has of the environment, which makes the consequences of its actions difficult to predict.

To these, we add a third one that we think is particularly relevant for RL agents and cannot be satisfactorily described by the first two:

3. *Active risk*, which is due to randomization over actions purposefully practiced by the agent, typically as an exploration technique.

We have introduced this third kind of risk because, differently from inherent risk, it is avoidable, in the sense that the agent has direct control over its own degree of randomness; differently from epistemic risk, it is stochastic in a frequentist sense. Moreover, it allows to highlight a trade-off between epistemic and active risk, since random actions are a way to explore, collect more knowledge, and reduce epistemic risk. We further elaborate on this point in Chapter 8.

Another fundamental distinction pertains the temporal scope of safety concerns. In some cases, for instance when an agent is trained in simulation and then deployed in the actual environment, we only care about the safety of the final behavior. In most real-world applications, where learning takes part partially or entirely in a safety-critical environment, we also care about intermediate behavior. We follow Amodei et al. (2016) in using the term *Safe Exploration* to broadly indicate this latter problem of safety *during* the learning process. To avoid confusion, we call *constrained exploration* the problem of avoiding dangerous states, which constitutes an important special case (Section 4.6).

[1] Whether machine learning algorithms should be used at all in these delicate contexts is an open ethical issue that lies beyond the scope of this manuscript. See for instance Matthias (2004) on the *responsibility gap* and Bird et al. (2016) on the ethics of exploration.

Finally, we must treat irrecoverable damage with particular care. In the classic RL formulation, rewards are always additive. Even online-learning approaches, which are concerned with the performance of the agent during the learning process, allow to compensate large losses with large gains. In safety-critical settings, a large loss may simply not be tolerable. This motivates the monotonic-improvement approaches of Chapter 5.

In the next sections we will review the different ways in which safety can be enforced in a RL agent.

4.2 Agent Alignment

The first safety concern in training RL agents is that the reward signal correctly encodes the intended optimal behavior. This is a special case of the broad *artificial intelligence alignment* problem, which encompasses the technical (Amodei et al., 2016) and philosophical issues (Bostrom, 2017) of aligning machine behavior with human desires and well-being.

Let us focus on RL agents. First, the reward designer may have overlooked possible negative side effects that are compatible with, or even facilitate reward maximization. Even when all possible side effects have been taken into account, the agent may still find counterintuitive ways to maximize the received reward, for instance by exploiting bugs in the implementation, a phenomenon known as reward hacking (Amodei et al., 2016).

Besides careful engineering, a possible solution is to learn the reward function itself from human demonstrations, an approach known as *inverse reinforcement learning* (Ng and Russell, 2000). Alternatively, the reward signal can be replaced or reshaped by real-time human feedback (Knox and Stone, 2009). When this is not possible, training the agent on a set of similar tasks could help it develop a common-sense understanding of the intended optimal behavior (Agarwal et al., 2019).

4.3 Risk-averse Reinforcement Learning

According to the reward hypothesis (Section 2.1.4), we should be able to encode any concept of goal, including safety concerns, into a properly engineered reward signal. In practice, it can be difficult to formulate safety as the maximization of an expected return, since rare events can have large hazardous consequences regardless of their probability. Risk-averse RL approaches deal with the optimization of a modified objective function that more naturally captures the effects of risky behavior, typically by penalizing high-variance return distributions or conservatively focusing on worst-case scenarios. Some possible objectives are minimum reward (Heger, 1994), return mean-variance (Castro et al., 2012; Tamar and Mannor, 2013; Prashanth and Ghavamzadeh, 2014, 2016), Sharpe ratio (Moody and Saffell, 2001), utility functions (Moldovan and Abbeel, 2012a; Shen et al., 2014), Conditional Value at Risk (Morimura et al., 2010; Tamar et al., 2017; Chow et al., 2017), and the recently introduced *mean-volatility* (Bisi et al., 2020; Zhang et al., 2020b). In this framework,

the main challenge is to recover the theoretical guarantees and practical performance of RL algorithms originally designed for risk-neutral expected-return objectives. Historically, risk-averse approaches have been particularly popular in financial applications of RL (Moody and Saffell, 2001), where hazards mostly correspond to monetary losses.

4.4 Constrained Markov Decision Processes

In some applications, especially those that may threaten the safety of humans, hazardous behavior must be avoided at any cost. In such cases, explicitly describing unsafe behavior can be much easier than constructing a sufficiently risk-aware objective function. One of the most natural approaches to safe RL consists in adding explicit constraints to the MDP problem formulation. This was recently advocated as *the* standard approach by Amodei et al. (2016).

The *Constrained Markov Decision Process* (CMDP) formalism is the subject of a book by Altman (1999). It is obtained by equipping an MDP with a set of *cost functions* $C = \{c_i\}_{i=1}^{K}$. Each cost function can be regarded as a separate, additional (negated) reward signal and used to define a corresponding expected return $J_{c_i}(\pi)$ for each policy π. The set of *feasible policies* is defined as:

$$\Pi_C = \left\{\pi \in \Delta_{\mathcal{A}}^{S} \mid J_{c_i}(\pi) \leqslant 0 \text{ for } i = 1, \ldots, K\right\}. \tag{4.1}$$

For instance, if $f(\pi)$ measured the power consumption of a robot implementing policy π, a cost function $c_1(\pi) = f(\pi) - d$ could be used to exclude policies having a power consumption higher than some threshold d. The objective is to find the *feasible* policy maximizing expected return w.r.t. the original reward signal:

$$\max_{\pi \in \Pi_C} J(\pi). \tag{4.2}$$

The main approach to CMDPs is that of *Lagrange multipliers*, which reformulates the constrained problem as an unconstrained one with additional parameters (dual variables or multipliers). Introduced by Altman (1998) in the context of dynamic programming, it was first applied to policy optimization by Borkar (2005). The latter proposed an actor-critic algorithm that optimizes the Lagrangian while learning the multipliers via gradient descent on a lower time scale. See Bhatnagar and Lakshmanan (2012) and Tessler et al. (2019) for more recent improvements.

Lagrangian methods only guarantee the feasibility of the final policy, hence additional care is required to achieve safe exploration. Efroni et al. (2020a) study the exploration-exploitation dilemma in CMDPs, proposing an algorithm that optimistically updates both primal and dual variables. On the more practical side, Achiam et al. (2017) adopt a trust-region approach (Section 3.8) that also enforces the constraints during the learning process. Their CPO (Constrained Policy Optimization) algorithm is an extension of TRPO to the CMDP framework. Albeit CPO arguably represents the most practical and general RL algorithm for high-dimensional continuous control tasks with constraints, it shares with TRPO the concerns about the many approximations necessary for efficient implementation,

which make us question its validity as a safe algorithm. Indeed, approximations seem to sometimes invalidate safe exploration in practice, as suggested by the empirical evaluation by Amodei et al. (2016). This, if anything, serves as a reminder that the CMDP formalism is a powerful tool to *define* unsafe behavior, but *avoiding* it during the learning process requires careful algorithm design.

Another approach to CMDPs tries to translate the global constraints into local ones that can be more easily enforced (also during training). This can be done by introducing stricter, step-wise constraints, leading to conservative algorithms (Gábor et al., 1998; Chamie et al., 2016). More recently, Chow et al. (2018) proposed an approach based on Lyapunov functions, a powerful tool from control theory that can be used to translate global properties of a dynamical system into local ones. The main challenge here is to generate a Lyapunov function for the given MDP, for which Chow et al. (2018) provide a linear program. The Lyapunov function can then be used to turn RL algorithms into their safe counterpart with little effort. Applied to policy gradient algorithms, this approach is competitive with CPO on continuous control benchmarks (Chow et al., 2019).

4.5 Seldonian Reinforcement Learning

Let us now focus on the policy optimization problem:

$$\max_{\pi \in \Theta} J(\boldsymbol{\theta}), \tag{4.3}$$

where Θ is the set of candidate policy parameters and $J : \Theta \to \mathbb{R}$ is the usual performance measure, or expected return. Let $\mathfrak{f}$ denote a PO algorithm, $\mathcal{D}$ the data it collects and $\mathfrak{f}(\mathcal{D}) \in \Theta$ the resulting output, that is an approximate solution to (4.3). The CMDP formulation acts on the set of policies, restricting it via explicit constraints. Both risk-averse RL and agent alignment modify the performance measure: the first to encode risk aversion in the objective function, the second to ensure it is aligned with the needs of the user.

The *Seldonian machine learning* framework (Thomas et al., 2019) acts on a different, or complementary level, shifting the focus from the problem definition to the learning algorithm itself. Safety is encoded through a *desirability function* $g : \Theta \to \mathbb{R}$ which acts directly on the output $\mathfrak{f}(\mathcal{D})$ of the algorithm.[2] A safety condition in the Seldonian sense is then:

$$\mathbb{P}\left(g(\mathfrak{f}(\mathcal{D})) \geqslant 0\right) \geqslant 1 - \delta, \tag{4.4}$$

where the probability in the left-hand side is over possible datasets and the stochasticity of the algorithm itself, and δ is some acceptable failure probability. This shifts the responsibility of safe behavior from the user (or the person responsible of defining J and Θ for a specific problem) to the designer of the general-purpose learning algorithm. This framework is very general and can be also applied to supervised learning problems, where Θ is a set of feasible solutions and J an objective function.

[2]The general formulation by Thomas et al. (2019) admits multiple desirability functions, each with a corresponding failure probability. We consider a single function for simplicity of exposition.

For instance, if $\mathcal{D}$ is a dataset of images and $\mathfrak{f}$ a classification problem, J could measure the fraction of correctly classified instances, while g could encode some notion of fairness (e.g., Hardt et al., 2016).

4.5.1 Offline Seldonian RL

Early examples of Seldonian algorithms can be found in batch (or offline) RL, where the agent cannot interact with the environment and has only access to a dataset $\mathcal{D}$ of trajectories (or transitions). In this setting, the problem of *safety w.r.t. a baseline* consists in never (with high probability) producing policies whose expected return is under a user-defined threshold $J_0 \in \mathbb{R}$. Depending on the task and on how the reward signal is defined, such a low performance may correspond to unsafe behavior or just deemed unacceptable by the user. Often J_0 corresponds to the performance of a *baseline policy* π_0, i.e., $J_0 = J(\pi_0)$. In the Seldonian formulation, the desirability function is $g(\pi) = J(\pi) - J_0$. The High Confidence Policy Improvement algorithm by Thomas et al. (2015b,a) guarantees this form of safety by identifying good candidate policies with high predicted performance, but only returning them if they pass a statistical test based on importance-weighted performance estimates. This approach requires knowledge of the behavioral policies that generated the trajectories in $\mathcal{D}$. Ghavamzadeh et al. (2016) adopt a *robust* model-based approach, where the performance of candidate policies is evaluated on the worst possible model of the environment that is compatible with the available data. Laroche et al. (2019) introduce the idea of *baseline bootstrapping*: the produced policy is just a return-maximizing one (obtained with a batch RL algorithm such as Fitted Q-Iteration (Ernst et al., 2005)), but it is replaced by the baseline policy π_0 in all state-action pairs for which the available data are not enough to guarantee safe behavior with sufficient probability. Here it is assumed that the data in $\mathcal{D}$ have been collected with π_0. A *soft* variant is also possible where the output policy can disagree with the baseline on all state-action pairs, but only proportionally to the corresponding uncertainty (Nadjahi et al., 2019). See also Simão and Spaan (2019) for an extension to factored environments.

4.5.2 Online Seldonian RL

Online RL algorithms produce sequences of policies, each of which interacts with the environment, hence should meet safety requirements. From the Seldonian perspective, we are facing a sequence of safety conditions like (4.4). In particular, the natural online extension of safety w.r.t. a baseline is *monotonic improvement*, which requires each new policy to be at least as good as the previous one (or not significantly worse). Importantly, Garcelon et al. (2020) show that this kind of constraint does not prevent efficient exploration in principle.

Consider the sequence of policies $\pi_1, \pi_2, \ldots$ generated by an on-policy RL algorithm. Each new solution π_{k+1} is obtained from data collected with the previous policy π_k, which represents the current baseline. A desirability function $g(\pi_{k+1}) = J(\pi_{k+1}) - J(\pi_k)$ yields the following monotonic improvement condition:

$$J(\pi_{k+1}) \geqslant J(\pi_k), \tag{4.5}$$

which is required to hold with probability at least $1 - \delta$ for each $k = 0, 1, 2, \ldots$. This amounts to enforcing a monotonic improvement of the performance over time, avoiding oscillating trends. This *policy oscillation* problem was already known from the literature on approximate dynamic programming (Bertsekas, 2011; Wagner, 2011). In their seminal work on CPI, Kakade and Langford (Conservative Policy Iteration, 2002) provide the first monotonic improvement guarantee for RL algorithms. They consider policies that are mixtures of previous policies, recursively defined as:

$$\pi_{k+1}(a|s) = (1 - \alpha)\pi_k(a|s) + \alpha\pi_k^+(a|s), \tag{4.6}$$

where π_k^+ represents an (approximate) greedy policy improvement w.r.t. π_k and $\alpha \in [0, 1]$ is a step size. For $\alpha < 1$, Equation 4.6 is a conservative policy update since it combines the approximately greedy solution with the current baseline π_k. Kakade and Langford (2002, Theorem 4.1) show that, for this kind of update:

$$J(\pi_{k+1}) - J(\pi_k) \geqslant \frac{\alpha}{(1-\gamma)} \mathop{\mathbb{E}}_{\substack{s \sim d_\mu^{\pi_k} \\ a \sim \pi_k^+}} \left[A^{\pi_k}(s, a)\right] - \frac{2\alpha^2\gamma\epsilon}{(1-\gamma)^2(1-\alpha)}, \tag{4.7}$$

where $\epsilon = \max_{s \in \mathcal{S}} \left| \mathbb{E}_{a \sim \pi_k^+(s,a)} \left[A^{\pi_k}(s, a)\right] \right|$. The first term is the expected advantage of the new policy w.r.t. the old one by neglecting the effects of the update on the state distribution, and should be positive if the greedy policy is properly estimated. The negative part is a worst-case penalty that accounts for the aforementioned distributional mismatch. The purpose of the conservative update (4.6) is precisely to control the penalty by not fully trusting the promised advantage of the greedy policy. In fact, the step size α can be selected to guarantee monotonic performance improvement (Kakade and Langford, 2002, Corollary 4.2). A heuristic version of CPI for deep neural policies has been proposed by Vieillard et al. (2020).

Pirotta et al. (2013b, Theorem 3.5, Corollary 3.6) provide a more general performance improvement bound that holds for any pair of stationary policies π and π':

$$J(\pi') - J(\pi) \geqslant \frac{1}{(1-\gamma)} \mathop{\mathbb{E}}_{\substack{s \sim d_\mu^{\pi} \\ a \sim \pi'}} \left[A^{\pi}(s, a)\right] - \frac{\gamma\widetilde{\epsilon}}{2(1-\gamma)^2} \left\|\pi' - \pi\right\|_\infty \tag{4.8}$$

$$\geqslant \frac{1}{(1-\gamma)} \mathop{\mathbb{E}}_{\substack{s \sim d_\mu^{\pi} \\ a \sim \pi'}} \left[A^{\pi}(s, a)\right] - \frac{\gamma \left\|Q^\pi\right\|_\infty}{2(1-\gamma)^2} \left\|\pi' - \pi\right\|_\infty^2, \tag{4.9}$$

where $\widetilde{\epsilon} = \sup_{s,s' \in \mathcal{S}} \left| \mathbb{E}_{a \sim \pi'(\cdot|s')} \left[A^{\pi}(s', a)\right] - \mathbb{E}_{a \sim \pi'(\cdot|s)} \left[A^{\pi}(s, a)\right] \right|$. Besides leading to tighter monotonic improvement guarantees for the conservative update (4.6), this bound characterizes the pessimistic penalty for a generic policy update in terms of the dissimilarity of the two policies (measured by the supremum norm in this case).

For *stochastic* policies, Schulman et al. (2015a, Theorem 1) independently prove

the following bound:

$$J(\pi') - J(\pi) \geqslant \frac{1}{(1-\gamma)} \mathop{\mathbb{E}}_{\substack{s\sim d_\mu^\pi \\ a\sim\pi'}} \left[A^\pi(s,a)\right] - \frac{4\gamma \left\|A^\pi\right\|_\infty}{(1-\gamma)^2} \left(\sup_{s\in\mathcal{S}} \mathcal{TV}(\pi(\cdot|s), \pi'(\cdot|s))\right)^2. \tag{4.10}$$

where $\mathcal{TV}(p,q) = \frac{1}{2}\left\|p-q\right\|_1$ denotes the total variation distance of probability measures p, q. A similar bound could be obtained from (4.9) by upper-bounding the supremum norm $\left\|\pi' - \pi\right\|_\infty$ with $\sup_{s\in\mathcal{S}} \left\|\pi'(\cdot|s) - \pi(\cdot|s)\right\|_1 = 2\sup_{s\in\mathcal{S}} \mathcal{TV}(\pi(\cdot|s), \pi'(\cdot|s))$:

$$J(\pi') - J(\pi) \geqslant \frac{1}{(1-\gamma)} \mathop{\mathbb{E}}_{\substack{s\sim d_\mu^\pi \\ a\sim\pi'}} \left[A^\pi(s,a)\right] - \frac{2\gamma \left\|Q^\pi\right\|_\infty}{(1-\gamma)^2} \left(\sup_{s\in\mathcal{S}} \mathcal{TV}(\pi(\cdot|s), \pi'(\cdot|s))\right)^2. \tag{4.11}$$

An alternative bound where the total variation is not squared could be similarly obtained from (4.8):

$$J(\pi') - J(\pi) \geqslant \frac{1}{(1-\gamma)} \mathop{\mathbb{E}}_{\substack{s\sim d_\mu^\pi \\ a\sim\pi'}} \left[A^\pi(s,a)\right] - \frac{2\gamma\epsilon}{(1-\gamma)^2} \sup_{s\in\mathcal{S}} \mathcal{TV}(\pi(\cdot|s), \pi'(\cdot|s)), \tag{4.12}$$

where $\epsilon = \sup_{s\in\mathcal{S}} \left|\mathbb{E}_{a\sim\pi'(\cdot|s)}\left[A^\pi(s,a)\right]\right|$. Achiam et al. (2017) provide a tighter version of the latter:

$$J(\pi') - J(\pi) \geqslant \frac{1}{(1-\gamma)} \mathop{\mathbb{E}}_{\substack{s\sim d_\mu^\pi \\ a\sim\pi'}} \left[A^\pi(s,a)\right] - \frac{2\gamma\epsilon}{(1-\gamma)^2} \mathop{\mathbb{E}}_{s\sim d_\mu^\pi} \left[\mathcal{TV}(\pi(\cdot|s), \pi'(\cdot|s))\right], \tag{4.13}$$

where the supremum over states on the total variation is replaced by an expectation.

Both Schulman et al. (2015a) and Achiam et al. (2017) suggest to upper bound the total variation with the KL divergence for practical purposes. This, applied to policy gradient updates of parametric policies, was the original motivation of the TRPO algorithm, and its CMDP extension CPO. We have already discussed the questionable safety of these methods in Sections 3.8 and 4.4. Previously, Pirotta et al. (2013a) had already adapted (4.9) to the special case of shallow Gaussian policies, deriving a policy gradient algorithm with monotonic improvement guarantees. We discuss these and derived safe policy gradient methods in more depth in Chapter 5.

An alternative approach is that by Cohen et al. (2018), who propose a variant of high confidence policy improvement (Thomas et al., 2015a) that repeatedly updates a whole set of behavioral policies. Enforcing *diversity* of these policies is a form of efficient exploration that is compatible with the safety constraint. Cohen et al. (2019) apply this approach to policy gradients.

4.6 Constrained Exploration

Within artificial intelligence safety, a peculiar challenge of RL agents is to collect the data necessary for learning in a safe way. This problem is not naturally captured by

the Seldonian machine learning framework (4.4), which considers a fixed dataset $\mathcal{D}$. In Section 4.5.2, we have extended it to consider a sequence of interactions. However, in off-policy online RL, the data-collecting (behavioral) policy could be different from the output of the learning algorithm (the target policy), which is at the same time a challenge and an opportunity for safe behavior. An important branch of safe RL deals with the problem of explicitly avoiding *unsafe states* (or state-action combinations) during the learning process. In this framework, safe exploration is the problem of collecting the data necessary for learning while avoiding the unsafe states. As mentioned, the wording *safe exploration* is often used in a narrower sense to denote precisely this problem (Pecka and Svoboda, 2014). We use *constrained exploration* to avoid confusion, since we intend safe exploration as the more general problem of avoiding hazards during the learning process.

Let us first see how the safe RL approaches presented in the previous sections can be applied to constrained exploration. When the behavioral policy used to interact with the environment coincides with the target policy optimized by the learning algorithm (or is a slightly modified version, as in DDPG), we can encode the unsafety of some states in the form of large negative rewards. This allows to use a standard RL algorithm, the main disadvantage being that the agent must first discover unsafe states by visiting them, which could lead to early termination in some applications. With some additional prior knowledge on the task, we could construct an appropriate safety function, to be employed by a Seldonian RL algorithm, penalizing policies that may lead to unsafe states. Similarly, the CMDP formalism can be adapted to constrained exploration by making the constraints state-wise instead of trajectory wise. The latter is the approach adopted by Dalal et al. (2018), who add a constraint-enforcing *safety layer* to parametric policies learned via policy gradient.

However, we can gain more control on the safety of our agent by explicitly designing a safe exploration process. The approach by Hans et al. (2008) is based on two components: a safety function that measures the degree of safety of a state, and a backup policy that can always lead the agent back to a safe state. Both may be unknown in practice. Some works bypass this problem using *human intervention*, either by integrating RL with human demonstrations (e.g., Abbeel et al., 2010), letting the agent explicitly ask for advice to a human *teacher* while it explores (e.g., Clouse, 1997), or having the teacher intervene whenever it thinks the agent is in danger (e.g., Clouse and Utgoff, 1992). See García and Fernández (2015) for a survey, and Plisnier et al. (2018, 2019) for recent safe policy gradients algorithms leveraging human advice.

In many other applications, the agent can only rely on a small amount of prior knowledge and must learn the safety function from very same data it is trying to safely collect. Turchetta et al. (2016, 2019) devise a way, based on Gaussian processes, to learn the safety function from experience data without ever visiting unsafe states (with high probability). Starting from an initial set of knowingly safe states and a Gaussian-process prior on the safety function, their SafeMDP algorithm gradually expands the safe set with states that are predicted to be safe,

concurrently improving the safety function estimate with new data.[3]

The concept of *ergodicity* is fundamental for constrained exploration. Informally, an MDP is ergodic if every state can be eventually reached from any other state by following an appropriate policy. Without this property, the agent could get stuck in *traps*: regions of the state space from which it is impossible to escape and that may inexorably lead to unsafe situations. Unfortunately, most real problems are not ergodic. Moldovan and Abbeel (2012b) propose to restrict the set of feasible policies to those that preserve ergodicity with high probability, and formalize this problem as a CMDP. The SafeMDP agent by Turchetta et al. (2016) can also preserve ergodicity by avoiding states where no safe action is available. A similar approach, based on Lyapunov functions, is used by Berkenkamp et al. (2017) to guarantee stability of the controlled system, and is successfully applied to the safe control of quadcopters (Berkenkamp et al., 2016).

We refer the reader to the dissertation by Berkenkamp (2019) for a recent overview of constrained exploration, including applications to robotics.

4.7 Our Perspective on Safe Reinforcement Learning

Our brief review of safe RL approaches should have reinforced the idea that safety is a complex and multi-faceted problem. Contrarily to Amodei et al. (2016), we do not advocate for a specific solution concept. Agent alignment and risk-aversion help define meaningful objectives; the CMDP formulation allows to easily exclude unsafe policies; Seldonian machine learning can be used to ensure that learned policies meet the safety requirements; and constrained exploration guarantees that data are collected in a safe way.[4] Hence, we believe all of these perspective should play a role in the design of truly safe learning agents for real-world applications.

However, this dissertation deals with a particular aspect of safety and with a specific class of learning algorithms. First, we are interested in the behavior of agents *during* the learning process, which places our work within the scope of the safe exploration problem, broadly intended. Second, we adopt the Seldonian RL approach, and focus on how the uncertain nature of the learning process itself can produce undesired behavior. To decouple this problem from the one of defining safe behavior, we assume that safety requirements can be encoded in the standard expected-return objective with no additional costs or constraints. This is the same approach adopted in the works on *safety w.r.t. a baseline* and on *monotonic performance improvement* presented in Section 4.5. The underlying assumption may be too strong for most real-world applications. However, it facilitates the study of safe policy optimization from a theoretical perspective. We think this allows to tackle some fundamental issues of learning in the real world and can lead to valuable insights and techniques that can be later integrated with other, complementary safety measures (e.g., Bisi et al., 2020).

[3] The ability of Gaussian Processes to model their own uncertainty plays a fundamental role in making well-informed predictions on the safety of unknown states. The agent starts very cautious and becomes more confident as it collects more data.

[4] Some works already try to combine the different approaches. For instance, in (Bisi et al., 2020) we study monotonic improvement of a risk-averse objective.

Third, we restrict our attention to policy gradient algorithms. These are the most promising RL approaches for real-world control tasks, which are also the most affected by safety concerns. The policy-based approach also allows to restrict the set of feasible policies without affecting the learning algorithm, which is a convenient way to exploit prior domain knowledge to rule out some unsafe behaviors from the start.

The original contributions we present in the next chapters are all concerned with ensuring safety in the sense that we have delineated here. Chapters 5 and 8 are specifically on monotonic improvement guarantees for policy gradient methods, the latter also addressing *safe adaptive stochasticity*, a particular aspect of safe exploration that arises when the agent has control of the amount of *dithering* (random action perturbation). The other contributions deal with the tension between safety and efficiency. In Chapter 6, we study the risks associated with re-using data multiple times for the sake of efficiency. In Chapter 7, we study the convergence properties of policy gradient algorithms with a focus on sample complexity.

Monotonic Improvement Guarantees 5

In this chapter, we present our approach to policy optimization with monotonic improvement guarantees. Our theoretical and algorithmic contributions are specifically taylored to policy gradient algorithms. The approach is similar to (Pirotta et al., 2013a) and (Papini et al., 2017), but is not limited to Gaussian policies. Compared to more general works on monotonic improvement (Schulman et al., 2015a), our approach allows to design safe algorithms that can actually be executed without compromises even in the continuous MDP setting.

We start by formally defining the monotonic improvement problem and its basic assumptions in Section 5.1. In Section 5.2, we introduce *smoothing policies*, a family of stochastic policies with favorable properties for policy optimization that includes commonly used policy classes. In Section 5.3, we show that the performance of smoothing policies is a smooth function of the policy parameters. This property sets the stage for monotonic improvement theory, but will also prove useful for convergence guarantees in Chapter 7. In Section 5.4, we show how adaptively selecting the step size of an actor-only policy gradient algorithm is enough to guarantee monotonic improvement. When policy gradients need to be estimated from data, the batch size also plays a fundamental role (Papini et al., 2017). After characterizing the variance of policy gradient estimators in Section 5.5, we propose an adaptive schedule for the batch size that guarantees monotonic improvement under constant step size. We call this algorithm *Safe Policy Gradient* (SPG), presented in Section 5.6. In Section 5.7, we compare our theoretical results on monotonic improvement with related ones from the literature. Besides the theoretical interest, our approach can be over-conservative in practice due to worst-case assumptions. In Section 5.8, we propose a slightly heuristic variant of SPG, named *Semi-Safe Policy Gradient* (SSPG), which relies on a Gaussianity assumption to actively measure the gradient estimation error. In Section 5.9, we provide an empirical evaluation of the proposed algorithms. Finally, we discuss the theory-practice gap in Section 5.10.

This chapter is based on ongoing work on safe policy gradients of which a

preprint (Papini et al., 2019b) is available.

5.1 Problem Definition

Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma, \mu \rangle$ and a parametric policy class $\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} \in \Delta_{\mathcal{A}}^{\mathcal{S}} \mid \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^d\}$, we want to design a policy optimization algorithm that guarantees:

$$J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant 0, \tag{5.1}$$

for each iteration $k \geqslant 0$, where $\boldsymbol{\theta}_0$ are the initial policy parameters and the algorithm selects $\boldsymbol{\theta}_{k+1}$ based on data collected with $\pi_{\boldsymbol{\theta}_k}$ via direct interaction with the environment. As discussed in Section 4.5, this *Monotonic Improvement* (MI) property ensures safe behavior provided the reward function encodes all sources of risk and the initial policy is safe by design. As we will see, this is a rather strict requirement. A first relaxation is to require that (5.1) holds *with high probability*:

$$\mathbb{P}\left(J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant 0 \mid \boldsymbol{\theta}_k\right) \geqslant 1 - \delta, \tag{5.2}$$

for some small *failure probability* $\delta > 0$.[1] Further relaxations are discussed in Section 3.12.

In the following sections, we will construct a policy gradient algorithm that guarantees (5.2). We will obtain it as a variant of Actor-only PG (Algorithm 2) by a conservative, adaptive choice of two fundamental meta-parameters:

- ◇ The step size α of the parameter updates, which controls how long a step we make in the direction of the policy gradient. Long steps are risky given the local nature of the first-order update, which ignores the curvature of the objective function. To this, we must add the fact that the actual direction used to update the policy parameters is just a noisy estimate of the true gradient. On the other hand, a larger step size can yield faster convergence to an optimum.

- ◇ The batch size N, that is the number of trajectories used to estimate the policy gradient at each iteration. A larger N yields a more accurate estimation, at the cost of interacting with the environment for a larger number of episodes per policy update.

The two meta-parameters have a strong relationship: a larger batch size yields a more reliable gradient direction, allowing a larger step size. For this reason, we will select them jointly. Moreover, given the first-order and stochastic nature of the gradient update, their effect strongly depends on the local properties of $\boldsymbol{\theta}_k$ and the data $\mathcal{D}_k$ collected with $\pi_{\boldsymbol{\theta}_k}$. For this reason, we will make both a function of the most recent policy parameters and collected trajectories. Finally, the both engage a

[1] The probability in (5.2) is over $\boldsymbol{\theta}_{k+1}$, which is a random variable since it is computed from data sampled with $\pi_{\boldsymbol{\theta}_k}$. More rigorously, the probability should be conditioned on a filtration representing all randomness prior to the k-th iteration.

trade-off between safety and learning speed. To fully account for this, we will select α_k and N_k as to maximize the following objective (Papini et al., 2017):

$$\alpha_k, N_k = \arg \max_{\alpha \geqslant 0, N \in \mathbb{N}} \Upsilon(\alpha, N; \boldsymbol{\theta}_k, \mathcal{D}_k) = \frac{J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k)}{N_k}, \tag{5.3}$$

where $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \widehat{\nabla} J(\boldsymbol{\theta}_k; \mathcal{D}_k)$. Intuitively, Υ is the average performance improvement per collected trajectory. Note that $\Upsilon \geqslant 0$ implies MI. Since computing Υ in a reliable way is typically not feasible, we resort to maximizing a lower bound B_δ such that:

$$\Upsilon(\alpha, N; \boldsymbol{\theta}_k, \mathcal{D}_k) \geqslant \frac{B_\delta(\alpha, N; \boldsymbol{\theta}_k, \mathcal{D}_k)}{N_k} \qquad \text{w.p. at least } 1 - \delta. \tag{5.4}$$

This is a conservative approach since a non-negative value of B_δ still implies (5.2).

We would like our MI guarantee to hold for the most general class of MDPs and policies. If we restrict our attention to stochastic policies (like Schulman et al., 2015a), we only need a boundedness assumption on the reward:

Assumption 5.1 *The reward function r is uniformly bounded, in absolute value, as $\|r\|_\infty \leqslant R_{\max} < \infty$,*

and no regularity assumptions on the transition kernel. That is, we can still guarantee monotonic improvement in environments with very irregular, even discontinuous rewards and transitions. That is due to the *smoothing effect* of stochastic policies, which is better characterized in the next section. Under Assumption 5.1, the value functions are uniformly bounded for any policy π as follows:

$$\|V^\pi\|_\infty \leqslant \|Q^\pi\|_\infty \leqslant \frac{R_{\max}}{1 - \gamma}, \tag{5.5}$$

where the first inequality follows from $V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)} \left[Q^\pi(s, a) \right]$ and the second one from $Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^\infty \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a \right]$ and Holder's inequality.

5.2 Smoothing Policies

We introduce a family of parametric stochastic policies having properties that we deem desirable for policy-gradient learning. Besides the MI guarantees that are the focus of this chapter, they are also the natural subjects of the kind of convergence analysis presented in Chapter 7. We call them *smoothing*, as they induce the smoothness of the performance regardless of the environment's regularity:

Definition 5.2.1 *Let $\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} \mid \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^m\}$ be a class of twice-differentiable parametric policies. We call it* smoothing *if the parameter space Θ is convex and there exist non-negative constants ξ_1, ξ_2, ξ_3 such that, for every state and in expectation over actions, the euclidean norm of the score function:*

$$\sup_{s \in \mathcal{S}} \mathbb{E}_{a \sim \pi_{\boldsymbol{\theta}}(\cdot|s)} \Big[\, \| \nabla \log \pi_{\boldsymbol{\theta}}(a|s) \| \, \Big] \leqslant \xi_1, \tag{5.6}$$

the squared euclidean norm of the score function:

$$\sup_{s\in\mathcal{S}} \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\Big[\, \|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\|^2 \,\Big] \leqslant \xi_2, \tag{5.7}$$

and the spectral norm of the observed information:

$$\sup_{s\in\mathcal{S}} \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\Big[\, \|\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s)\| \,\Big] \leqslant \xi_3, \tag{5.8}$$

are upper-bounded for all $\boldsymbol{\theta} \in \Theta$.

Note that the definition requires the bounding constants ξ_1, ξ_2, ξ_3 to be independent from the policy parameters and the state. For this reason, the existence of such constants depends on the policy parametrization. We call a policy class (ξ_1, ξ_2, ξ_3)-smoothing when we want to specify the bounding constants.

We now show that the most commonly used policy classes, the Gaussian for continuous actions and the Softmax for finite actions, are smoothing, and compute their smoothing constants (later summarized in Table 5.1). We also provide an example of non-smoothing stochastic policy, which raises a concern about adaptive exploration.

5.2.1 Gaussian policies

Let Π_Θ be the family of scalar-action, shallow Gaussian policies defined in Section 3.3.4, with $\Theta \subseteq \mathbb{R}^d$. Assume that the policy standard deviation σ is fixed and that the feature function $\boldsymbol{\phi}$ is bounded in *euclidean norm*, i.e., $\sup_{s\in\mathcal{S}} \|\boldsymbol{\phi}(s)\| \leqslant \phi_{\max} < \infty$. Then Π_Θ is (ξ_1, ξ_2, ξ_3)-smoothing with the following constants:

$$\xi_1 = \frac{2\phi_{\max}}{\sqrt{2\pi}\sigma}, \qquad\qquad \xi_2 = \xi_3 = \frac{\phi_{\max}^2}{\sigma^2}. \tag{5.9}$$

Proof Fix a $\boldsymbol{\theta} \in \Theta$. Let $x \equiv \frac{a - \boldsymbol{\theta}^T \boldsymbol{\phi}(s)}{\sigma}$. Note that $\mathcal{A} = \mathbb{R}$ and $\mathrm{d}a = \sigma\, \mathrm{d}x$. We use the scores and the observed information computed in Section 3.3.4, where $\boldsymbol{\theta} \equiv \boldsymbol{\theta}_\mu$ since σ is fixed. First, we compute ξ_1:

$$\begin{aligned}
\mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)} [\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\|] &= \int_{\mathbb{R}} \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2} \left\| \frac{\boldsymbol{\phi}(s)}{\sigma} x \right\| \sigma\, \mathrm{d}x \\
&\leqslant \frac{\phi_{\max}}{\sqrt{2\pi}\sigma} \int_{\mathbb{R}} e^{-x^2/2} |x|\, \mathrm{d}x \\
&= \frac{2\phi_{\max}}{\sqrt{2\pi}\sigma} := \xi_1.
\end{aligned} \tag{5.10}$$

Then, we compute ξ_2:

$$\begin{aligned}\mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\|^2\right] &= \int_{\mathbb{R}} \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2} \left\|\frac{\boldsymbol{\phi}(s)}{\sigma} x\right\|^2 \sigma \,\mathrm{d}x \\ &\leqslant \frac{\phi_{\max}^2}{\sqrt{2\pi}\sigma^2} \int_{\mathbb{R}} e^{-x^2/2} x^2 \,\mathrm{d}x \\ &= \frac{\phi_{\max}^2}{\sigma^2} := \xi_2. \end{aligned} \tag{5.11}$$

Finally, we compute ξ_3:

$$\begin{aligned}\mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\left\|\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s)\right\|\right] &= \int_{\mathbb{R}} \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2} \left\|\frac{\boldsymbol{\phi}(s)}{\sigma} x\right\|^2 \sigma \,\mathrm{d}x \\ &\leqslant \frac{\phi_{\max}^2}{\sqrt{2\pi}\sigma^2} \int_{\mathbb{R}} e^{-x^2/2} x^2 \,\mathrm{d}x \\ &= \frac{\phi_{\max}^2}{\sigma^2} := \xi_3. \end{aligned} \tag{5.12}$$

■

From (5.9), we can see that Gaussian policies are no longer smoothing if we let the standard deviation σ be a function of policy parameters. Indeed, all the smoothing constants tend to infinity as σ tends to zero. This is expected, since only stochastic policies can be smoothing. A simple workaround is to enforce a lower bound on $\sigma_{\boldsymbol{\theta}}$. However, we would like to converge to deterministic policies in practice. We discuss safe ways to learn the policy variance in Chapter 8, in the context of safe exploration.

5.2.2 Softmax policies

Let Π_Θ be the class of *linear* Softmax policies[2] described in Section 3.3.3. Assume that the feature function ϕ is bounded in *euclidean norm*, i.e., $\sup_{s\in\mathcal{S},a\in\mathcal{A}} \|\boldsymbol{\phi}(s,a)\| \leqslant \phi_{\max} < \infty$. Then, Π_Θ is (ξ_1,ξ_2,ξ_3)-smoothing with the following constants:

$$\xi_1 = \frac{2\phi_{\max}}{\tau}, \qquad \xi_2 = \frac{4\phi_{\max}^2}{\tau^2}, \qquad \xi_3 = \frac{2\phi_{\max}^2}{\tau^2}, \tag{5.13}$$

where $\tau > 0$ is the fixed temperature.

Proof In this case, we can simply bound $\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\|$ and $\left\|\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s)\right\|$ uniformly over states and actions. The smoothing conditions follow trivially. We use the expressions for the score and the observed information computed in Section 3.3.3.

[2]Extending the guarantees from (Papini et al., 2017) to Softmax policies was the original motivation that led to the general formulation we present in this chapter.

First, we compute ξ_1 and ξ_2:

$$\begin{aligned}\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\| &\leqslant \frac{1}{\tau}\left(\|\boldsymbol{\phi}(s,a)\| + \left\|\mathop{\mathbb{E}}_{a'\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\boldsymbol{\phi}(s,a')\right]\right\|\right) \\ &\leqslant \frac{2\phi_{\max}}{\tau}, \end{aligned} \tag{5.14}$$

hence $\sup_{s\in\mathcal{S}} \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\|\right] \leqslant \frac{2\phi_{\max}}{\tau} := \xi_1$ and $\sup_{s\in\mathcal{S}} \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\|^2\right] \leqslant \frac{4\phi^2_{\max}}{\tau^2} := \xi_2$.

Then, we compute ξ_3:

$$\begin{aligned}\left\|\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s)\right\| &\leqslant \frac{1}{\tau^2}\mathop{\mathbb{E}}_{a'\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\left\|\boldsymbol{\phi}(s,a')\left(\mathop{\mathbb{E}}_{a''\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\boldsymbol{\phi}(s,a'')\right] - \boldsymbol{\phi}(s,a')\right)^T\right\|\right] \\ &\leqslant \frac{1}{\tau^2}\mathop{\mathbb{E}}_{a'\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\left\|\boldsymbol{\phi}(s,a')\right\|\left\|\mathop{\mathbb{E}}_{a''\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\boldsymbol{\phi}(s,a'') - \boldsymbol{\phi}(s,a')\right]\right\|\right] \\ &\leqslant \frac{1}{\tau^2}\mathop{\mathbb{E}}_{a'\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\left\|\boldsymbol{\phi}(s,a')\right\|\mathop{\mathbb{E}}_{a''\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\left\|\boldsymbol{\phi}(s,a'')\right\| + \left\|\boldsymbol{\phi}(s,a')\right\|\right]\right] \\ &\leqslant \frac{2\phi^2_{\max}}{\tau^2}, \end{aligned} \tag{5.15}$$

hence $\sup_{s\in\mathcal{S}} \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\left\|\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s)\right\|\right] \leqslant \frac{2\phi^2_{\max}}{\tau^2} := \xi_3$. ■

Note the similarity with the Gaussian constants from (5.9). The temperature parameter τ plays a similar role to the standard deviation σ. However, as mentioned, we do not loose any generality by setting τ to a fixed value.

5.3 Smooth Policy Optimization

In this section we show that, for smoothing policies, the objective of the policy optimization problem is indeed a *smooth* function of policy parameters, that is:

$$\left\|\nabla J(\boldsymbol{\theta}') - \nabla J(\boldsymbol{\theta})\right\| \leqslant L\left\|\boldsymbol{\theta}' - \boldsymbol{\theta}\right\|, \tag{5.16}$$

for every $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta$, where $L > 0$ is a finite constant. In particular, when (5.16) holds, we say $J(\boldsymbol{\theta})$ is *L-smooth*. Equivalently, the policy gradient is L-Lipschitz continuous under the Euclidean metric. This property of the objective function is often assumed in non-convex optimization, and will play an important role in establishing convergence rates in Chapter 7.

We prove this key result by showing that the *policy Hessian* $\nabla^2 J(\boldsymbol{\theta})$ for a smoothing policy has bounded spectral norm. First, we write the policy Hessian for a general parametric policy in the following convenient form:

Lemma 5.1 *(Kakade, 2001b, equation 6) Given a twice-differentiable parametric policy $\pi_{\boldsymbol{\theta}}$, the policy Hessian is:*

$$\nabla^2 J(\boldsymbol{\theta}) = \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s\sim d^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}} \Big[\nabla \log \pi_{\boldsymbol{\theta}}(a|s) \nabla^T Q^{\boldsymbol{\theta}}(s,a) + \nabla Q^{\boldsymbol{\theta}}(s,a) \nabla^T \log \pi_{\boldsymbol{\theta}}(a|s) + \left(\nabla \log \pi_{\boldsymbol{\theta}}(a|s) \nabla^T \log \pi_{\boldsymbol{\theta}}(a|s) + \nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s)\right) Q^{\boldsymbol{\theta}}(s,a)\Big].$$

Proof The first derivation was provided in (Kakade, 2001b), we restate it for the sake of clarity. We first compute the Hessian of the state-value function:

$$\begin{aligned}
\nabla^2 V^{\boldsymbol{\theta}}(s) &= \nabla^2 \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\, \mathrm{d}a \\
&= \nabla \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \left[\nabla^T \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a) + \nabla^T Q^{\boldsymbol{\theta}}(s,a)\right] \mathrm{d}a \qquad (5.17) \\
&= \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \left[\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a) + \nabla \log \pi_{\boldsymbol{\theta}}(a|s) \nabla^T Q^{\boldsymbol{\theta}}(s,a) \right. \\
&\quad \left. + \nabla Q^{\boldsymbol{\theta}}(s,a) \nabla^T \log \pi_{\boldsymbol{\theta}}(a|s) + \nabla^2 Q^{\boldsymbol{\theta}}(s,a)\right] \mathrm{d}a \qquad (5.18) \\
&= \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \Bigg[\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a) + \nabla \log \pi_{\boldsymbol{\theta}}(a|s) \nabla^T Q^{\boldsymbol{\theta}}(s,a) \\
&\quad + \nabla Q^{\boldsymbol{\theta}}(s,a) \nabla^T \log \pi_{\boldsymbol{\theta}}(a|s) \\
&\quad + \nabla^2 \left(r(s,a) + \gamma \int_{\mathcal{S}} p(s'|s,a) V^{\boldsymbol{\theta}}(s')\, \mathrm{d}s' \right) \Bigg] \mathrm{d}a \qquad (5.19) \\
&= \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a|s) \left[\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a) + \nabla \log \pi_{\boldsymbol{\theta}}(a|s) \nabla^T Q^{\boldsymbol{\theta}}(s,a) \right. \\
&\quad \left. + \nabla Q^{\boldsymbol{\theta}}(s,a) \nabla^T \log \pi_{\boldsymbol{\theta}}(a|s)\right] \mathrm{d}a + \gamma \int_{\mathcal{S}} p_{\boldsymbol{\theta}}(s'|s) \nabla^2 V^{\boldsymbol{\theta}}(s')\, \mathrm{d}s' \\
&= g(s) + \frac{\gamma}{1-\gamma} \int_{\mathcal{S}} d_s^{\boldsymbol{\theta}}(s') g(s')\, \mathrm{d}s', \qquad (5.20)
\end{aligned}$$

where

$$\begin{aligned}
g(s) = &\left(\nabla \log \pi_{\boldsymbol{\theta}}(a|s) \nabla^T \log \pi_{\boldsymbol{\theta}}(a|s) + \nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s)\right) Q^{\boldsymbol{\theta}}(s,a) \\
&+ \nabla \log \pi_{\boldsymbol{\theta}}(a|s') \nabla^T Q^{\boldsymbol{\theta}}(s,a) + \nabla Q^{\boldsymbol{\theta}}(s,a) \nabla^T \log \pi_{\boldsymbol{\theta}}(a|s),
\end{aligned}$$

(5.17) is from the log trick, (5.18) is from another application of the log trick, (5.19) is from (2.4), and (5.20) is from Lemma 2.1 with $\nabla^2 V^{\boldsymbol{\theta}}(s')$ as the recursive term. Computing the Hessian of the performance is then trivial:

$$\nabla^2 J(\boldsymbol{\theta}) = \nabla^2 \int_{\mathcal{S}} \mu(s) V^{\boldsymbol{\theta}}(s)\, \mathrm{d}s = \int_{\mathcal{S}} \mu(s) \nabla^2 V^{\boldsymbol{\theta}}(s)\, \mathrm{d}s, \qquad (5.21)$$

where the first equality is from (3.1). Combining (5.20), (5.21) and (2.15) we obtain the statement of the lemma. ■

We can now bound the policy Hessian for a smoothing policy:

Lemma 5.2 *Given a (ξ_1, ξ_2, ξ_3)-smoothing policy $\pi_{\boldsymbol{\theta}}$, the spectral norm of the policy Hessian can be upper-bounded as follows:*

$$\left\|\nabla^2 J(\boldsymbol{\theta})\right\| \leqslant \frac{R_{\max}}{(1-\gamma)^2}\left(\frac{2\gamma\xi_1^2}{1-\gamma} + \xi_2 + \xi_3\right).$$

Proof We start by bounding the gradient of the value function (see Equation 3.55) in Euclidean norm:

$$\begin{aligned}
\left\|\nabla V_{\boldsymbol{\theta}}(s)\right\| &\leqslant \mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\left\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\right\|\right] \\
&\quad + \frac{\gamma}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s'\sim d_s^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s')}}\left[\left\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s') Q^{\boldsymbol{\theta}}(s',a)\right\|\right] \\
&\leqslant \frac{R_{\max}}{1-\gamma} \mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\left\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\right\|\right] \\
&\quad + \frac{\gamma R_{\max}}{(1-\gamma)^2} \mathop{\mathbb{E}}_{\substack{s'\sim d^{\boldsymbol{\theta}}(\cdot|s) \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s')}}\left[\left\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s')\right\|\right] \qquad (5.22) \\
&\leqslant \frac{\xi_1 R_{\max}}{(1-\gamma)^2}, \qquad (5.23)
\end{aligned}$$

where (5.22) is from the Cauchy-Schwarz inequality and (5.5), and (5.22) is from the smoothing assumption. Next, we bound the gradient of the quality function. From (3.52):

$$\begin{aligned}
\left\|\nabla Q^{\boldsymbol{\theta}}(s,a)\right\| &\leqslant \gamma \mathop{\mathbb{E}}_{s'\sim p(\cdot|s,a)}\left[\left\|V^{\boldsymbol{\theta}}(s)\right\|\right] \\
&\leqslant \frac{\gamma\xi_1 R_{\max}}{(1-\gamma)^2}, \qquad (5.24)
\end{aligned}$$

where (5.24) is from (5.23). Finally, from Lemma 5.1:

$$
\begin{aligned}
(1-\gamma)\left\|\nabla^2 J(\boldsymbol{\theta})\right\| \leqslant & \mathop{\mathbb{E}}_{\substack{s\sim d^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}}\left[\left\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\nabla^T Q^{\boldsymbol{\theta}}(s,a)\right\|\right] \\
& + \mathop{\mathbb{E}}_{\substack{s\sim d^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}}\left[\left\|\nabla Q^{\boldsymbol{\theta}}(s,a)\nabla^T \log \pi_{\boldsymbol{\theta}}(a|s)\right\|\right] \\
& + \mathop{\mathbb{E}}_{\substack{s\sim d^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}}\left[\left\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\nabla^T \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\right\|\right] \\
& + \mathop{\mathbb{E}}_{\substack{s\sim d^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}}\left[\left\|\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\right\|\right] \qquad (5.25) \\
\leqslant 2 & \mathop{\mathbb{E}}_{\substack{s\sim d^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}}\left[\left\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\right\| \left\|\nabla Q^{\boldsymbol{\theta}}(s,a)\right\|\right] \\
& + \mathop{\mathbb{E}}_{\substack{s\sim d^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}}\left[\left\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\right\|^2 \left|Q^{\boldsymbol{\theta}}(s,a)\right|\right] \\
& + \mathop{\mathbb{E}}_{\substack{s\sim d^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}}\left[\left\|\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s)\right\| \left|Q^{\boldsymbol{\theta}}(s,a)\right|\right] \qquad (5.26) \\
\leqslant & \frac{2\gamma\xi_1 R_{\max}}{(1-\gamma)^2} \mathop{\mathbb{E}}_{\substack{s\sim d^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}}\left[\left\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\right\|\right] \\
& + \frac{R_{\max}}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s\sim d^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}}\left[\left\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\right\|^2\right] \\
& + \frac{R_{\max}}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s\sim d^{\boldsymbol{\theta}} \\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}}\left[\left\|\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s)\right\|\right] \qquad (5.27) \\
\leqslant & \frac{R_{\max}}{(1-\gamma)}\left(\frac{2\gamma\xi_1^2}{1-\gamma} + \xi_2 + \xi_3\right), \qquad (5.28)
\end{aligned}
$$

where (5.25) is from Jensen inequality (all norms are convex) and the triangle inequality, (5.26) is from $\left\|\boldsymbol{x}\boldsymbol{y}^T\right\| = \|\boldsymbol{x}\| \|\boldsymbol{y}\|$ for any two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, (5.27) is from (5.5) and (5.24), and the last inequality is from the smoothing assumption. ■

The key result on the performance measure now follows immediately from a well-known characterization of smooth functions (see Appendix A):

Theorem 5.1 *Given a (ξ_1, ξ_2, ξ_3)-smoothing policy class Π_Θ, the performance $J(\boldsymbol{\theta})$ is L-smooth with the following Lipschitz constant:*

$$
L = \frac{R_{\max}}{(1-\gamma)^2}\left(\frac{2\gamma\xi_1^2}{1-\gamma} + \xi_2 + \xi_3\right). \qquad (5.29)
$$

Proof From Lemma 5.2, L is a bound on the spectral norm of the policy Hessian. From Lemma A.1, this is a valid Lipschitz constant for the policy gradient, hence

the performance is L-smooth. ■

The smoothness of the performance, in turn, yields the following property on the guaranteed performance improvement, which is the policy gradient version of a well known quadratic bound for smooth optimization:

Theorem 5.2 *Let Π_Θ be a (ξ_1, ξ_2, ξ_3)-smoothing policy class. For every $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta$:*

$$J(\boldsymbol{\theta}') - J(\boldsymbol{\theta}) \geqslant \langle \Delta\boldsymbol{\theta}, \nabla J(\boldsymbol{\theta}) \rangle - \frac{L}{2} \|\Delta\boldsymbol{\theta}\|^2,$$

where $\Delta\boldsymbol{\theta} = \boldsymbol{\theta}' - \boldsymbol{\theta}$ and $L = \frac{R_{\max}}{(1-\gamma)^2}\left(\frac{2\gamma\xi_1^2}{1-\gamma} + \xi_2 + \xi_3\right)$.

Proof It suffices to apply Lemma A.2 with the Lipschitz constant from Theorem 5.1. ■

In the following sections, we will exploit this property of smoothing policies to enforce safety guarantees on the policy updates performed by Algorithm 2, i.e., stochastic gradient ascent updates. However, Theorem 5.2 applies to any policy update $\Delta\boldsymbol{\theta} \in \mathbb{R}^d$ as long as $\boldsymbol{\theta} + \Delta\boldsymbol{\theta} \in \Theta$.

5.4 Adaptive Step Size

From now on, we will focus on a (ξ_1, ξ_2, ξ_3)-smoothing policy class Π_Θ with $\Theta \subseteq \mathbb{R}^d$. We first focus on the problem of selecting the best adaptive step size. In our final algorithm, described in Section 5.6, we will optimize the step size and the batch size jointly.

5.4.1 Exact framework

We first consider an ideal setting where we have access to a *First-order Oracle* (FO), that is, we can obtain the exact policy gradient $\nabla J(\boldsymbol{\theta})$ for any parameter vector $\boldsymbol{\theta} \in \Theta$. This assumption is clearly not realistic, and will be removed in Section 5.6. In this simplified framework, monotonic performance improvement can be guaranteed *deterministically*, as in (5.1). Moreover, the only relevant meta-parameter is the step size α_k of the update.

We first need a computable lower bound on the performance improvement:

Theorem 5.3 *Let Π_Θ be a (ξ_1, ξ_2, ξ_3)-smoothing policy class. Let $\boldsymbol{\theta}_k \in \Theta$ and $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \nabla J(\boldsymbol{\theta}_k)$, where $\alpha_k > 0$. Provided $\boldsymbol{\theta}_{k+1} \in \Theta$, the performance improvement of $\boldsymbol{\theta}_{k+1}$ w.r.t. $\boldsymbol{\theta}_k$ can be lower-bounded as follows:*

$$J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant \alpha \|\nabla J(\boldsymbol{\theta}_k)\|^2 - \alpha^2 \frac{L}{2} \|\nabla J(\boldsymbol{\theta}_k)\|^2 := B(\alpha; \boldsymbol{\theta}_k),$$

where $L = \frac{R_{\max}}{(1-\gamma)^2}\left(\frac{2\gamma\xi_1^2}{1-\gamma} + \xi_2 + \xi_3\right)$.

Proof This is just a special case of Theorem 5.2 with $\Delta\boldsymbol{\theta} = \alpha_k \nabla J(\boldsymbol{\theta}_k)$. ■

This bound is in the typical form of performance improvement bounds discussed in Section 4.5.2: a positive term accounting for the anticipated advantage of $\boldsymbol{\theta}_{k+1}$ over $\boldsymbol{\theta}_k$, and a penalty term accounting for the mismatch between the two policies, which makes the anticipated advantage less reliable. In our case, the mismatch is measured by the curvature of the performance w.r.t. the policy parameters, via the Lipschitz constant L of the policy gradient. This lower bound is quadratic in α_k, hence we can easily find the optimal step size α_k^*.

Corollary 5.4 *Let $B(\cdot; \boldsymbol{\theta}_k)$ be the guaranteed performance improvement of an exact policy gradient update, as defined in Theorem 5.3. Under the same assumptions, $B(\cdot; \boldsymbol{\theta}_k)$ is maximized for all $k \geqslant 0$ by the constant step size $\alpha^* = 1/L$, which guarantees the following non-negative performance improvement:*

$$J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant \frac{\|\nabla J(\boldsymbol{\theta}_k)\|^2}{2L}.$$

Proof We just maximize $B(\alpha; \boldsymbol{\theta}_k)$, which is a quadratic function of α. The global optimum $B(\alpha^*; \boldsymbol{\theta}_k) = \frac{\|\nabla J(\boldsymbol{\theta}_k)\|^2}{2L}$ is attained by $\alpha^* = \frac{1}{L}$. The improvement guarantee follows from Theorem 5.3. ■

Note that in the exact framework a constant step size is enough to guarantee MI. This is not the case when the gradient needs to be estimated from data.

5.4.2 Approximate Framework

In practice, we cannot compute the exact gradient $\nabla J(\boldsymbol{\theta}_k)$ given the partial information available to the agent. Instead, we assume to have access to a *First-Order Stochastic Oracle* (FSO), that is, a policy gradient estimator $\widehat{\nabla} J(\boldsymbol{\theta}; \mathcal{D})$, where $\mathcal{D}$ is a dataset of N trajectories. We require it satisfies the following assumptions:

Assumption 5.2 (Unbiasedness) *For all $\boldsymbol{\theta} \in \Theta$:*

$$\mathop{\mathbb{E}}_{\mathcal{D} \sim p_{\boldsymbol{\theta}}} \left[\widehat{\nabla} J(\boldsymbol{\theta}; \mathcal{D}) \right] = \nabla J(\boldsymbol{\theta}).$$

Assumption 5.3 (Bounded error) *For every $\delta \in (0, 1)$ there exists a non-negative constant ϵ_δ such that, with probability at least $1 - \delta$:*

$$\left\| \nabla J(\boldsymbol{\theta}) - \widehat{\nabla} J(\boldsymbol{\theta}; \mathcal{D}) \right\| \leqslant \frac{\epsilon_\delta}{\sqrt{N}}, \tag{5.30}$$

for all $\boldsymbol{\theta} \in \Theta$, where $N = |\mathcal{D}|$.

The first assumption is easily satisfied by the unbiased policy gradient estimators presented in Section 3.5. The second assumption is a high-probability upper bound

on the gradient estimation error, and is also realistic (Pirotta et al., 2013a), as we will remark in Section 5.5.

Under these assumptions, we can adapt Theorem 5.3 to the stochastic gradient case as follows:

Theorem 5.5 *Let Π_Θ be a (ξ_1, ξ_2, ξ_3)-smoothing policy class. Let $\boldsymbol{\theta}_k \in \Theta \subseteq \mathbb{R}^d$ and $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k v_k$, where $\alpha_k \geqslant 0$, $v_k = \widehat{\nabla} J(\boldsymbol{\theta}_k; \mathcal{D}_k)$, $\mathcal{D}_k \sim p_{\boldsymbol{\theta}_k}$, and $\widehat{\nabla} J$ satisfies Assumptions 5.2 and 5.3. Let $N = |\mathcal{D}_k| \geqslant 1$. Provided $\boldsymbol{\theta}_{k+1} \in \Theta$, the performance improvement of $\boldsymbol{\theta}_{k+1}$ w.r.t. $\boldsymbol{\theta}_k$ can be lower bounded, with probability at least $1 - \delta$, as follows:*

$$\begin{aligned} J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) &\geqslant \alpha_k \left(\|v_k\| - \frac{\epsilon_\delta}{\sqrt{N}} \right) \max \left\{ \|v_k\|, \frac{\|v_k\| + \frac{\epsilon_\delta}{\sqrt{N}}}{2} \right\} \\ &\quad - \frac{\alpha^2 L}{2} \|v_k\|^2 \\ &\coloneqq B_\delta(\alpha_k, N; \boldsymbol{\theta}_k, \mathcal{D}_k), \end{aligned} \tag{5.31}$$

where $L = \frac{R_{\max}}{(1-\gamma)^2} \left(\frac{2\gamma\xi_1^2}{1-\gamma} + \xi_2 + \xi_3 \right)$.

Proof From Assumption 5.3, with probability at least $1 - \delta$:

$$\begin{aligned} \|\nabla J(\boldsymbol{\theta}_k)\| &\geqslant \|v_k\| - \|\nabla J(\boldsymbol{\theta}_k) - v_k\| \\ &\geqslant \|v_k\| - \frac{\epsilon_\delta}{\sqrt{N}}, \end{aligned} \tag{5.32}$$

thus:

$$\|\nabla J(\boldsymbol{\theta}_k)\|^2 \geqslant \max \left\{ \|v_k\| - \frac{\epsilon_\delta}{\sqrt{N}}, 0 \right\}^2. \tag{5.33}$$

Then, from the law of cosines:

$$\begin{aligned} \langle v_k, \nabla J(\boldsymbol{\theta}_k) \rangle &= \frac{1}{2} \left(\|v_k\|^2 + \|\nabla J(\boldsymbol{\theta}_k)\|^2 - \|\nabla J(\boldsymbol{\theta}_k) - v_k\|^2 \right) \\ &\geqslant \frac{1}{2} \left(\|v_k\|^2 + \max \left\{ \|v_k\| - \frac{\epsilon_\delta}{\sqrt{N}}, 0 \right\}^2 - \frac{\epsilon_\delta^2}{N} \right) \end{aligned} \tag{5.34}$$

where (5.34) is from (5.33) and Assumption 5.3. We first consider the case in which $\|v_k\| > \frac{\epsilon_\delta}{\sqrt{N}}$:

$$\begin{aligned} \langle v_k, \nabla J(\boldsymbol{\theta}_k) \rangle &\geqslant \frac{1}{2} \left(\|v_k\|^2 + \left(\|v_k\| - \frac{\epsilon_\delta}{\sqrt{N}} \right)^2 - \frac{\epsilon_\delta^2}{N} \right) \\ &= \left(\|v_k\| - \frac{\epsilon_\delta}{\sqrt{N}} \right) \|v_k\|. \end{aligned}$$

Then, we consider the case in which $\|v_k\| \leqslant \frac{\epsilon_\delta}{\sqrt{N}}$:

$$\begin{aligned}\langle v_k, \nabla J(\boldsymbol{\theta}_k)\rangle &\geqslant \frac{1}{2}\left(\|v_k\|^2 - \frac{\epsilon_\delta^2}{N}\right)\\ &= \left(\|v_k\| - \frac{\epsilon_\delta}{\sqrt{N}}\right)\frac{\|v_k\| + \frac{\epsilon_\delta}{\sqrt{N}}}{2}.\end{aligned}$$

The two cases can be unified as follows:

$$\langle v_k, \nabla J(\boldsymbol{\theta}_k)\rangle \geqslant \left(\|v_k\| - \frac{\epsilon_\delta}{\sqrt{N}}\right)\max\left\{\|v_k\|, \frac{\|v_k\| + \frac{\epsilon_\delta}{\sqrt{N}}}{2}\right\}. \tag{5.35}$$

From Theorem 5.2 with $\Delta\boldsymbol{\theta} = \alpha v_k$ we obtain, with probability at least $1-\delta$:

$$\begin{aligned}J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) &\geqslant \langle \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k, \nabla J(\boldsymbol{\theta}_k)\rangle - \frac{L}{2}\left\|\boldsymbol{\theta}'_k - \boldsymbol{\theta}_k\right\|^2\\ &= \alpha\left\langle \widehat{\nabla}_N J(\boldsymbol{\theta}_k), \nabla J(\boldsymbol{\theta}_k)\right\rangle - \frac{\alpha^2 L}{2}\left\|\widehat{\nabla}_N J(\boldsymbol{\theta}_k)\right\|^2\\ &\geqslant \alpha\left(\|v_k\| - \frac{\epsilon_\delta}{\sqrt{N}}\right)\max\left\{\|v_k\|, \frac{\|v_k\| + \frac{\epsilon_\delta}{\sqrt{N}}}{2}\right\}\\ &\quad - \frac{\alpha^2 L}{2}\left\|\widehat{\nabla}_N J(\boldsymbol{\theta}_k)\right\|^2,\end{aligned}$$

where the last inequality is from (5.35). ■

From Theorem 5.5 we can easily obtain an optimal step size, as done in the exact setting, provided the batch size is sufficiently large:

Corollary 5.6 *Let $B_\delta(\alpha_k, N; \boldsymbol{\theta}_k, \mathcal{D}_k)$ be guaranteed performance improvement of a stochastic policy gradient update $\alpha_k v_k$, as defined in Theorem 5.5, with $N = |\mathcal{D}_k|$. Under the same assumptions, provided the batch size satisfies the following constraint:*

$$N \geqslant \frac{\epsilon_\delta^2}{\|v_k\|^2}, \tag{5.36}$$

$B_\delta(\cdot, N; \boldsymbol{\theta}_k, \mathcal{D}_k)$ is maximized by the following adaptive step size:

$$\alpha_k^* = \frac{1}{L}\left(1 - \frac{\epsilon_\delta}{\sqrt{N}\,\|v_k\|}\right), \tag{5.37}$$

which guarantees, with probability at least $1-\delta$, the following non-negative performance improvement:

$$J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant \frac{\left(\|v_k\| - \frac{\epsilon_\delta}{\sqrt{N}}\right)^2}{2L}. \tag{5.38}$$

Proof Let $N_0 = \epsilon_\delta^2 / \|v_k\|^2$. When $N \leqslant N_0$, the second argument of the max operator in (5.31) is selected. In this case, no positive improvement can be guaranteed and the optimal non-negative step size is $\alpha = 0$. Thus, we focus on the case $N > N_0$. In this case, the first argument of the max operator is selected. Optimizing B_δ as a function of α alone, which is again quadratic, yields (5.37) as the optimal step size and (5.38) as the maximum guaranteed improvement. Note that the latter is always non-negative under the batch-size assumption (5.36). ■

In this case, the optimal step size is adaptive, i.e., time-varying and data-dependent. As suggested by intuition, a larger batch size, by making the gradient estimate more reliable, allows to use a larger step size. The constant, optimal step size for the exact case (Corollary 5.4) is recovered in the limit of infinite data, i.e., $N \to \infty$.

5.5 Variance of Policy Gradient Estimators

Before completing the design of our SPG algorithm with batch-size selection (Section 5.6), we provide bounds on the estimation error ϵ_δ involved in Theorem 5.5 for common policy gradient estimators. One way to characterize ϵ_δ, is to upper-bound the variance of the estimator. With *variance* of a vector $\boldsymbol{v}$, we always mean the following:

$$\mathbb{V}\mathrm{ar}(\boldsymbol{v}) = \mathrm{Tr}\left(\mathbb{C}\mathrm{ov}(\boldsymbol{v}, \boldsymbol{v})\right) = \mathbb{E}\left[\left\|\boldsymbol{v} - \mathbb{E}\left[\boldsymbol{v}\right]\right\|^2\right]. \tag{5.39}$$

The following result, based on Chebyshev's inequality, allows to obtain an ϵ_δ satisfying Assumption 5.3 from a variance upper bound:

Lemma 5.3 *Let $\widehat{\nabla} J$ be an* unbiased *estimator of ∇J such that:*

$$\underset{\mathcal{D} \sim p_{\boldsymbol{\theta}}}{\mathbb{V}ar}\left[\widehat{\nabla} J(\boldsymbol{\theta}; \mathcal{D})\right] \leqslant \frac{\nu^2}{N},$$

where $N = |\mathcal{D}|$. Then $\widehat{\nabla} J$ satisfies Assumption 5.3 with $\epsilon_\delta = \nu / \sqrt{\delta}$.

Proof We apply the vector version of Chebyshev's inequality (Ferentios, 1982). ■

In the remaining of this section, we provide upper bounds on the variance of the REINFORCE and G(PO)MDP estimators, generalizing existing results for Gaussian policies (Zhao et al., 2011; Pirotta et al., 2013a) to smoothing policies.

5.5.1 Variance of REINFORCE

Assume we are dealing with an episodic task. For simplicity, take Assumption 2.1 to hold (the bias this may introduce is discussed in Section 2.5.2). We begin by bounding the variance of the REINFORCE estimator (3.76):

Lemma 5.4 *Given a (ξ_1, ξ_2, ξ_3)-smoothing policy class Π_Θ and a task horizon H, for every $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^d$, the variance of the REINFORCE estimator (with zero baseline) is upper-bounded as follows:*

$$\underset{\mathcal{D}\sim p_{\boldsymbol{\theta}}}{\mathbb{V}ar}\left[\widehat{\nabla}_R J(\boldsymbol{\theta};\mathcal{D})\right] \leqslant \frac{H\xi_2 R_{\max}^2(1-\gamma^H)^2}{N(1-\gamma)^2},$$

where $N = |\mathcal{D}|$.

Proof Let $g_{\boldsymbol{\theta}}(\tau) \coloneqq \left(\sum_{t=0}^{H-1}\gamma^t r(a_t,s_t)\right)\left(\sum_{t=0}^{H-1}\nabla\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\right)$ with $s_t, a_t \in \tau$ for $t = 0,\dots,H-1$. Using the definition of REINFORCE (3.76):

$$\begin{aligned}
\underset{\mathcal{D}\sim p_{\boldsymbol{\theta}}}{\mathbb{V}ar}\left[\widehat{\nabla}_R J(\boldsymbol{\theta};\mathcal{D})\right] &= \frac{1}{N}\underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathbb{V}ar}\left[g_{\boldsymbol{\theta}}(\tau)\right] \\
&\leqslant \frac{1}{N}\underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathbb{E}}\left[\|g_{\boldsymbol{\theta}}(\tau)\|^2\right] \\
&\leqslant \frac{R_{\max}^2(1-\gamma^H)^2}{N(1-\gamma)^2}\underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathbb{E}}\left[\left\|\sum_{t=0}^{H-1}\nabla\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\right\|^2\right] \\
&\leqslant \frac{R_{\max}^2(1-\gamma^H)^2}{N(1-\gamma)^2}\sum_{i=1}^{d}\underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathbb{E}}\left[\sum_{t=0}^{H-1}\left(\nabla_{\theta_i}\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\right)^2\right. \\
&\quad \left. + 2\sum_{t=0}^{H-2}\sum_{h=t+1}^{H-1}\nabla_{\theta_i}\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\nabla_{\theta_i}\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right] \\
&= \frac{R_{\max}^2(1-\gamma^H)^2}{N(1-\gamma)^2}\underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathbb{E}}\left[\sum_{t=0}^{H-1}\|\nabla\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\|^2\right] \qquad (5.40) \\
&= \frac{R_{\max}^2(1-\gamma^H)^2}{N(1-\gamma)^2}\sum_{t=0}^{H-1}\underset{s_0\sim\mu}{\mathbb{E}}\left[\dots\underset{a_t\sim\pi_{\boldsymbol{\theta}}(\cdot|s_t)}{\mathbb{E}}\left[\|\nabla\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\|^2\,\middle|\,s_t\right]\dots\right] \\
&\leqslant \frac{H\xi_2 R_{\max}^2(1-\gamma^H)^2}{N(1-\gamma)^2},
\end{aligned}$$

where (5.40) is from the following:

$$\begin{aligned}
&\underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathbb{E}}\left[\sum_{t=0}^{H-2}\sum_{h=t+1}^{H-1}\nabla_{\theta_i}\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\nabla_{\theta_i}\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right] \\
&\quad = \sum_{t=0}^{H-2}\underset{s_0\sim\mu}{\mathbb{E}}\left[\dots\underset{a_t\sim\pi_{\boldsymbol{\theta}}(\cdot|s_t)}{\mathbb{E}}\left[\nabla_{\theta_i}\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\right.\right. \\
&\qquad \left.\left.\sum_{h=t+1}^{H-1}\underset{s_{t+1}\sim p(\cdot|s_t,a_t)}{\mathbb{E}}\left[\dots\underset{a_h\sim\pi_{\boldsymbol{\theta}}(\cdot|s_h)}{\mathbb{E}}\left[\nabla_{\theta_i}\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\,|\,s_h\right]\dots\,\middle|\,a_t\right]\,\middle|\,s_t\right]\dots\right] \\
&\quad = 0,
\end{aligned}$$

where the last equality is from (3.7). ■

This is a generalization of Lemma 5.3 from Pirotta et al. (2013a), which in turn is an adaptation of Theorem 2 from Zhao et al. (2011). In the Gaussian case, the original lemma is recovered by plugging the smoothing constant $\xi_2 = \phi_{\max}^2/\sigma^2$ from (5.9). Note also that, from the definition of smoothing policy, only the second condition (5.7) is actually necessary for Lemma 5.4 to hold.

5.5.2 Variance of G(PO)MDP

For the G(PO)MDP estimator (3.77), we obtain an upper bound that does not grow linearly with the horizon H:

Lemma 5.5 *Given a (ξ_1, ξ_2, ξ_3)-smoothing policy class Π_Θ and a task horizon H, for every $\boldsymbol{\theta} \in \Theta$, the variance of the G(PO)MDP estimator (with zero baseline) is upper-bounded as follows:*

$$\underset{\mathcal{D}\sim p_{\boldsymbol{\theta}}}{\mathbb{V}ar}\left[\widehat{\nabla}_G J(\boldsymbol{\theta};\mathcal{D})\right] \leqslant \frac{\xi_2 R_{\max}^2\Big(1-\gamma^H\Big)}{N(1-\gamma)^3},$$

where $N = |\mathcal{D}|$.

Proof Let $g_{\boldsymbol{\theta}}(\tau) := \sum_{t=0}^{H-1}\gamma^t r(a_t,s_t)\left(\sum_{h=0}^{t}\nabla\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right)$ with $s_t, a_t \in \tau$ for $t = 0,\dots,H-1$. Using the definition of G(PO)MDP (3.77):

$$\begin{aligned}
\underset{\mathcal{D}\sim p_{\boldsymbol{\theta}}}{\mathbb{V}ar}\left[\widehat{\nabla}_N J(\boldsymbol{\theta})\right] &= \frac{1}{N}\underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathbb{V}ar}\left[\sum_{t=0}^{H-1}\gamma^t r(a_t,s_t)\left(\sum_{h=0}^{t}\nabla\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right)\right] && (5.41)\\
&\leqslant \frac{1}{N}\underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathbb{E}}\left[\left(\sum_{t=0}^{H-1}\gamma^{t/2} r(a_t,s_t)\gamma^{t/2}\left(\sum_{h=0}^{t}\nabla\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right)\right)^2\right]\\
&\leqslant \frac{1}{N}\underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathbb{E}}\left[\left(\sum_{t=0}^{H-1}\gamma^t r(a_t,s_t)^2\right)\left(\sum_{t=0}^{H-1}\gamma^t\left(\sum_{h=0}^{t}\nabla\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right)^2\right)\right] && (5.42)\\
&\leqslant \frac{R_{\max}^2(1-\gamma^H)}{N(1-\gamma)}\underset{\tau\sim p_{\boldsymbol{\theta}}}{\mathbb{E}}\left[\sum_{t=0}^{H-1}\gamma^t\left(\sum_{h=0}^{t}\nabla\log\pi_{\boldsymbol{\theta}}(a_h|s_h)\right)^2\right]\\
&\leqslant \frac{\xi_2 R_{\max}^2(1-\gamma^H)}{N(1-\gamma)}\sum_{t=0}^{H-1}\gamma^t(t+1) && (5.43)\\
&= \frac{\xi_2 R_{\max}^2(1-\gamma^H)}{N(1-\gamma)^3}\left[1 - H\Big(\underbrace{\gamma^H-\gamma^{H+1}}_{\geqslant 0}\Big) - \gamma^H\right] && (5.44)\\
&\leqslant \frac{\xi_2 R_{\max}^2\Big(1-\gamma^H\Big)}{N(1-\gamma)^3},
\end{aligned}$$

where (5.41) is from the fact that the trajectories are i.i.d., (5.42) is from the Cauchy-Schwarz inequality, (5.43) is from the same argument used for (5.40) in the proof of Lemma 5.4, and (5.44) is from the sum of the arithmetico-geometric sequence. ■

This is a generalization of Lemma 5.5 from Pirotta et al. (2013a). Again, in the Gaussian case, the original lemma is recovered by plugging the smoothing constant $\xi_2 = \phi_{\max}/\sigma^2$ from (5.9). Note that this variance upper bound stays finite in the limit $H \to \infty$, which is not the case for REINFORCE.

The variance upper bounds of Lemma 5.4 and 5.5 are still valid if one uses the variance minimizing baselines by Peters and Schaal (2008b) presented in Section 3.5.4. This is because the zero baseline is a valid choice in the variance-minimization problems they are solving. It is an open problem whether tighter variance upper bounds can be derived in that case.

Thanks to Lemma 5.3, the variance upper bounds allow to characterize the estimation error ϵ_δ in Assumption 5.3. This, in turn, allows to specialize Theorem 5.5 to the REINFORCE and the GPOMDP policy gradient estimator, respectively. Table 5.2 reports the value of ϵ_δ to be used in the different cases.

5.6 Safe Policy Gradient

In this section, we present our SPG algorithm that guarantees MI (5.2) at each step k with a user-defined failure probability of δ. It is a variant of actor-only PG (Algorithm 2) where the step size and the batch size are adaptive and are conservatively taken to maximize a lower bound on the per-trajectory performance improvement (5.3). The requirements are that the policy class is smoothing and that the gradient estimators is unbiased (Assumption 5.2) and has bounded error with probability $1 - \delta$ (Assumption 5.3).

We use Theorem 5.5 to find the jointly optimal step size and batch size, similarly to what was done by Papini et al. (2017) for Gaussian policies:

Corollary 5.7 *Let $B_\delta(\alpha_k, N_k; \boldsymbol{\theta}_k, \mathcal{D}_k)$ be the lower bound on the performance improvement of a stochastic policy gradient update $\alpha_k v_k$, as defined in Theorem 5.5. Under the same assumptions, the continuous relaxation of $B_\delta(\alpha_k, N_k; \boldsymbol{\theta}_k, \mathcal{D}_k)/N_k$ is maximized by the following constant step size and adaptive batch size:*

$$\begin{cases} \alpha^* = \frac{1}{2L} \\ N_k^* = \frac{4\epsilon_\delta^2}{\|v_k\|^2}. \end{cases} \tag{5.45}$$

Using α^* as the step size and any natural $N_k \geqslant N_k^*$ as the batch size in the stochastic gradient ascent update guarantees, with probability at least $1 - \delta$, the

following non-negative performance improvement:

$$J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant \frac{\|v_k\|^2}{8L}. \tag{5.46}$$

Proof Let $\widetilde{\Upsilon}(\alpha, N) = B_\delta(\alpha, N; \boldsymbol{\theta}_k, \mathcal{D}_k)/N$ for short and $N_0 = \epsilon_\delta^2 / \|v_k\|^2$. We consider the continuous relaxation of $\widetilde{\Upsilon}(\alpha, N)$, where N can be any positive real number. For $N \geqslant N_0$, the first argument of the max operator in (5.31) can be selected. Note that the second argument is always a valid choice, since it is a lower bound on the first one for every $N \geqslant 1$. Thus, we separately solve the following constrained optimization problems:

$$\begin{cases} \max_{\alpha, N} \frac{1}{N} \left(\alpha \|v_k\| \left(\|v_k\| - \frac{\epsilon_\delta}{\sqrt{N}} \right) - \alpha^2 \frac{L}{2} \|v_k\|^2 \right) \\ \text{s.t.} \quad \alpha \geqslant 0, \\ \qquad N > N_0, \end{cases} \tag{5.47}$$

and:

$$\begin{cases} \max_{\alpha, N} \frac{1}{N} \left(\frac{\alpha}{2} \left(\|v_k\|^2 - \frac{\epsilon_\delta^2}{N} \right) - \alpha^2 \frac{L}{2} \|v_k\|^2 \right) \\ \text{s.t.} \quad \alpha \geqslant 0, \\ \qquad N > 0. \end{cases} \tag{5.48}$$

Both problems can be solved in closed form using KKT conditions. The first one (5.47) yields $\widetilde{\Upsilon}^* = \left\| \widehat{\nabla}_N J(\boldsymbol{\theta}_k) \right\|^4 / \left(32 L \epsilon_\delta^2 \right)$ with the values of α^* and N_k^* given in (5.45). The second one (5.48) yields a worse optimum $\widetilde{\Upsilon}^* = \left\| \widehat{\nabla}_N J(\boldsymbol{\theta}_k) \right\|^4 / \left(54 L \epsilon_\delta^2 \right)$ with $\alpha = \frac{1}{3L}$ and $N = 3\epsilon_\delta^2 / \left\| \widehat{\nabla}_N J(\boldsymbol{\theta}_k) \right\|^2$. Hence, we keep the first solution. From Theorem 5.5, using α^* and N_k^* would guarantee $J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant \|v_k\|^2 / (8L)$. Of course, only integer batch sizes can be used. However, for $N \geqslant N_0$, the right-hand side of (5.31) is monotonically increasing in N. Since $N_k^* \geqslant N_0$, the guarantee (5.46) is still valid when α^* and any (natural) $N_k \geqslant N_k^*$ are employed in the stochastic gradient-ascent update. ■

In this case, the optimal step size is constant, and is exactly half the one for the exact case (Corollary 5.4). In turn, the batch size is adaptive: when the norm of the (estimated) gradient is small, a large batch size is selected. Intuitively, this allows to counteract the variance of the estimator, which is large relatively to the gradient magnitude. This also agrees with the intuition that the batch size should be increased as we approach convergence.

Note that the optimal batch size N_k^* from Corollary 5.7 cannot be computed in advance since it depends on the stochastic gradient v_k, which in turn should be computed using $N_k \geqslant N_k^*$ trajectories. We can break this circular dependence by collecting one trajectory at a time until the batch size is large enough.[3]

[3] A more rigorous analysis would require a martingale argument and, possibly, an adaptive confidence schedule for the inner loop of Algorithm 5.

Algorithm 5 SPG (Safe Policy Gradient)

Input: initial policy parameters $\boldsymbol{\theta}_0$, gradient Lipschitz constant L and error bound ϵ_δ
$\alpha = \frac{1}{2L}$
for $k = 0, 1, \dots$ **do**
    $\mathcal{D} = \varnothing$
    $N = 0$
    do
        Collect trajectory $\tau \sim p_{\boldsymbol{\theta}_k}$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{\tau\}$
        $N \leftarrow N + 1$
        $v_k = \widehat{\nabla} J(\boldsymbol{\theta}_k; \mathcal{D})$
    while $N < \frac{4\epsilon_\delta^2}{\|v_k\|^2}$
    $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha v_k$
end for

Our *Safe Policy Gradient* (SPG) algorithm is outlined in Algorithm 5. Besides guaranteeing MI, using the meta-parameters from Corollary 5.7, SPG removes the need for tuning, a task that is typically done by hand or through a time-consuming grid search (Duan et al., 2016). The gradient Lipschitz constant L can be retrieved from Table 5.1 depending on the policy class (Gaussian or Softmax). Similarly, the error upper bound ϵ_δ can be retrieved from Table 5.2 depending on the gradient estimator (REINFORCE or G(PO)MDP), the desired confidence $1 - \delta$, and the policy-dependent constant ξ_2, which is also reported in Table 5.1. The gradient estimate v_k must be re-computed for every new trajectory, or updated in an incremental fashion. The latter is non-trivial when variance-minimizing baselines (Peters and Schaal, 2008b) are employed. In practice, a batch version is also possible, where data collected at the k-th iteration are used to compute the optimal batch size for the next one. The empirical behavior should not differ from the one of Algorithm 5, provided the gradient magnitude does not change too abruptly. The main disadvantage of the batch version is that it requires the batch size for the first iteration to be manually selected.

5.7 Related Works

In this section we discuss previous results on MI guarantees for policy gradients. See Section 4.5.2 for a more general overview of monotonically improving RL.

Specific performance improvement bounds for policy gradient algorithms were first provided by Pirotta et al. (2013a) by adapting previous results on policy iteration (Pirotta et al., 2013b, in turn an improvement over the seminal work by Kakade and Langford (2002)) to continuous MDPs. However, the penalty term can only be computed for shallow Gaussian policies in practice. The bound for the

Table 5.1: Smoothing constants ξ_1, ξ_2, ξ_3 and policy-gradient Lipschitz constant L for Gaussian and Softmax policies, where $\phi_{\max}$ is an upper bound on the euclidean norm of the feature function, $R_{\max}$ is the maximum absolute-valued reward, γ is the discount factor, σ is the standard deviation of the Gaussian policy and τ is the temperature of the Softmax policy.

	Gaussian	**Softmax**
ξ_1	$\frac{2\phi_{\max}}{\sqrt{2\pi}\sigma}$	$\frac{2\phi_{\max}}{\tau}$
ξ_2	$\frac{\phi^2_{\max}}{\sigma^2}$	$\frac{4\phi^2_{\max}}{\tau^2}$
ξ_3	$\frac{\phi^2_{\max}}{\sigma^2}$	$\frac{2\phi^2_{\max}}{\tau^2}$
L	$\frac{2\phi^2_{\max}R_{\max}}{\sigma^2(1-\gamma)^2}\left(1+\frac{2\gamma}{\pi(1-\gamma)}\right)$	$\frac{2\phi^2_{\max}R_{\max}}{\tau^2(1-\gamma)^2}\left(3+\frac{4\gamma}{1-\gamma}\right)$

Table 5.2: Estimation error of the REINFORCE and GPOMDP policy gradient estimators on a single trajectory, where H is the task horizon, γ is the discount factor and $R_{\max}$ is the maximum reward. The smoothing constant ξ_2 depends on the policy class (cf. Table 5.1). The reported value is an upper bound on the actual error with probability at least $1-\delta$.

	REINFORCE	**GPOMDP**
ϵ_δ	$\frac{R_{\max}(1-\gamma^H)}{(1-\gamma)}\sqrt{\frac{H\xi_2}{\delta}}$	$\frac{R_{\max}}{(1-\gamma)}\sqrt{\frac{\xi_2(1-\gamma^H)}{\delta(1-\gamma)}}$

exact framework (FO) is:

$$J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant \alpha_k \left\|\nabla J(\boldsymbol{\theta}_k)\right\|^2 - \alpha_k^2 \frac{\phi^2_{\max}R_{\max}}{\sigma^2(1-\gamma)^2}\left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)}\right)\left\|\nabla J(\boldsymbol{\theta}_k)\right\|_1^2, \tag{5.49}$$

where $|\mathcal{A}|$ denotes the volume of the action space. From Table 5.1, our bound for the same setting is (Theorem 5.4):

$$J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant \alpha_k \left\|\nabla J(\boldsymbol{\theta}_k)\right\|^2 - \alpha_k^2 \frac{\phi^2_{\max}R_{\max}}{\sigma^2(1-\gamma)^2}\left(1 + \frac{2\gamma}{\pi(1-\gamma)}\right)\left\|\nabla J(\boldsymbol{\theta}_k)\right\|^2, \tag{5.50}$$

which has the same dependence on the step size, the policy standard deviation σ, the effective horizon $(1-\gamma)^{-1}$, the maximum reward $R_{\max}$ and the maximum feature norm $\phi_{\max}$. Besides being more general, our penalty term does not depend on the problematic $|\mathcal{A}|$ term (the action space is theoretically unbounded for Gaussian policies) and replaces the l_1 norm of (5.49) with the smaller l_2 norm. Due to the different constants, we cannot say our penalty is always smaller, but the change of norm could make a big difference in practice, especially for large parameter dimension d. Pirotta et al. (2013a) also study the approximate framework (FSO). However, albeit formulated in terms of the estimated gradient, their lower bound (Theorem 5.2) *still pertains exact policy gradient updates*, since $\boldsymbol{\theta}_{k+1}$ is defined as $\boldsymbol{\theta}_k + \alpha_k \nabla J(\boldsymbol{\theta}_k)$. This easy-to-overlook observation makes our Theorem 5.5 the first

formal monotonic improvement guarantee for stochastic policy gradient updates of the form $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \widehat{\nabla} J(\boldsymbol{\theta}_k)$. Pirotta et al. (2013a) use their results to design an adaptive step-size schedule for REINFORCE and G(PO)MDP, similarly to what we propose in Section 5.4 but limited to Gaussian policies. Papini et al. (2017) rely on the same improvement lower bound 5.49 to design an adaptive-batch size algorithm, the most similar to our SPG. Again, their monotonic improvement guarantees are limited to shallow Gaussian policies.

Another related family of performance improvement lower bounds, inspired once again by Kakade and Langford (2002), is that of TRPO. These are very general results that apply to arbitrary pairs of stochastic policies, although they are mostly used to construct policy gradient algorithms in practice. Specializing Theorem 1 by Schulman et al. (2015a) to our setting and applying the KL lower bound suggested by the authors we can get the following:

$$\begin{aligned} J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s\sim d_\mu^{\boldsymbol{\theta}_k} \\ a\sim\pi_{\boldsymbol{\theta}_{k+1}}}} \left[A^{\boldsymbol{\theta}_k}(s,a)\right] \\ - \frac{2\gamma R_{\max}}{(1-\gamma)^3} \max_{s\in\mathcal{S}} \left\{\mathcal{KL}(\pi_{\boldsymbol{\theta}_k}(\cdot|s) \| \pi_{\boldsymbol{\theta}_{k+1}})(\cdot|s)\right\}, \end{aligned} \tag{5.51}$$

where $\pi_{\boldsymbol{\theta}}$ is a stochastic policy. Unfortunately, the lower bound for a policy gradient update (exact or stochastic) cannot be computed exactly. Approximations can lead to very good practical algorithms such as TRPO, but not to actually implementable algorithms with rigorous monotonic improvement guarantees like our SPG. Achiam et al. (2017) and Pajarinen et al. (2019) are able to remove some approximations, but not all.[4] If we were to derive a computable worst-case lower bound starting from (5.51), we would get a result similar to (5.49). In fact, Pirotta et al. (2013a) explicitly upper-bound the KL divergence in their derivations, which is why the final result is limited to Gaussian policies. We overcome this difficulty by directly upper-bounding the curvature of the objective function (Lemma 5.2). Furthermore, Theorem 5.2 suggests that our theory is not limited to policy gradient updates. We consider arbitrary update directions in Section 8.1.3.

Finally, Pirotta et al. (2015) provide performance improvement lower bounds (Lemma 8) and adaptive-step algorithms for policy gradients under Lipschitz continuity assumptions on the MDP and the policy. Our assumptions on the environment are much weaker since we only require boundedness of the reward. Intuitively, stochastic policies *smooth out* the irregularities of the environment in computing expected return objectives. In turn, the results of Pirotta et al. (2015) also apply to deterministic policies.

[4] This is not a critique of the TRPO algorithm per se. Besides the celebrated empirical results, TRPO is also theoretically justified (Neu et al., 2017), only not as a monotonically improving gradient-descent algorithm (Shani et al., 2020).

5.8 Characterizing the Estimation Error

The version of SPG we presented in Section 5.6 requires an upper bound on the policy gradient estimator error ϵ_δ. We have shown in Section 5.5 how this can be obtained for the REINFORCE and the G(PO)MDP gradient estimators using Chebyshev's inequality. As already argued by Papini et al. (2017), using statistical inequalities based on the empirical variance of the gradient estimate can reduce the over-conservativeness of safe policy gradient algorithms. In this section, we provide a characterization of the gradient estimation error based on a Gaussianity assumption on $\widehat{\nabla} J(\boldsymbol{\theta}_k)$. The plausibility of this assumption relies on the Central Limit Theorem, hence is only justified by sufficiently large batch sizes. We follow Thomas et al. (2015a) in calling this a *semi-safe* approach, and name the variant of SPG that employs Gaussian confidence regions SSPG.

Since we care about the error magnitude of a d-dimensional random vector, we propose to employ *ellipsoidal* confidence regions. For any $\delta \in (0,1)$, let $\mathcal{E}_\delta$ be the following set:

$$\mathcal{E}_\delta = \left\{ \boldsymbol{x} \in \mathbb{R}^d : \left(\widehat{\nabla} J(\boldsymbol{\theta}_k) - \boldsymbol{x} \right)^T \widehat{\Sigma}_k^{-1} \left(\widehat{\nabla} J(\boldsymbol{\theta}_k) - \boldsymbol{x} \right) < \frac{Nd}{N-d} F_{1-\delta,d,N-d} \right\}, \quad (5.52)$$

where $\widehat{\Sigma}_k$ is the sample covariance of $\widehat{\nabla} J(\boldsymbol{\theta}_t)$ and $F_{1-\delta,m,N-m}$ is the quantile $(1-\delta)$ of the F-distribution with d and $n-d$ degrees of freedom. This set is centered in $\widehat{\nabla} J(\boldsymbol{\theta}_t)$ and is delimited by an ellipsoid. It is a standard result (Härdle and Simar, 2012) that, with probability $1-\delta$, the true gradient is contained in this region, i.e., $\mathbb{P}\left(\nabla J(\boldsymbol{\theta}_k) \in \mathcal{E}_\delta\right) = 1-\delta$.

Equivalently, the difference $\widehat{\nabla} J(\boldsymbol{\theta}_k) - \nabla J(\boldsymbol{\theta}_k)$ is contained in the following origin-centered ellipsoid:

$$\mathcal{E}_\delta = \left\{ \boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x}^T A_\delta \boldsymbol{x} = 1 \right\}, \quad (5.53)$$

where $A_\delta = \left(\frac{NdF_{1-\delta,d,N-d}}{N-d} \widehat{\Sigma}_k \right)^{-1}$. Thus, the estimation error $\left\| \widehat{\nabla} J(\boldsymbol{\theta}_k) - \nabla J(\boldsymbol{\theta}_k) \right\|$ cannot be larger than the largest semi-axis of $\mathcal{E}_\delta$. Simple algebraic computations yield the following:

$$\left\| \widehat{\nabla} J(\boldsymbol{\theta}_k) - \nabla J(\boldsymbol{\theta}_k) \right\| \leqslant \sqrt{\frac{NdF_{1-\delta,d,n-d} \left\| \widehat{\Sigma}_k \right\|}{N-d}} \coloneqq \epsilon_\delta, \quad (5.54)$$

with probability at least $1-\delta$, where $\left\| \widehat{\Sigma} \right\|$ denotes the spectral norm (i.e., the largest eigenvalue) of the sample covariance. The latter can be computed efficiently with the Lanczos method (Lanczos, 1950). The error upper bound ϵ_δ can be directly used in Algorithm 11. As mentioned, we call this semi-safe variant SSPG.

5.9 Empirical Evaluation

In this section we provide an empirical evaluation of the proposed safe policy gradient algorithms. The worst-case assumptions made to guarantee monotonic improvement

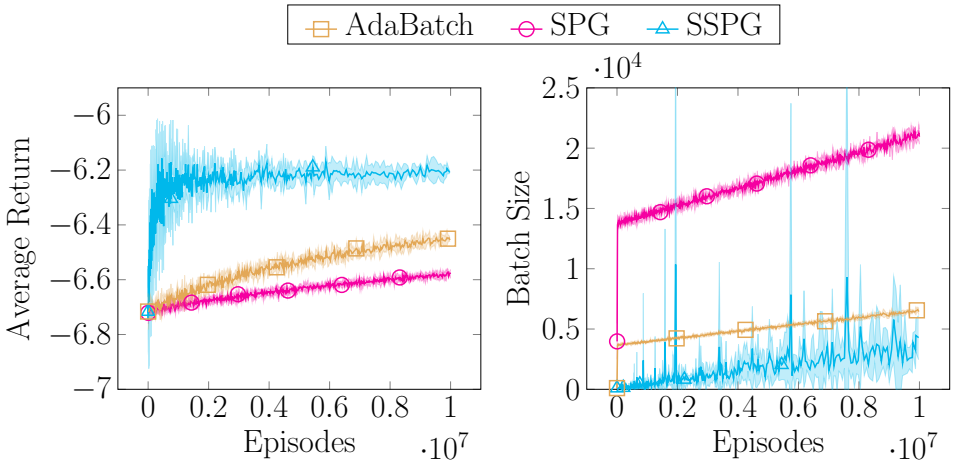


FIGURE 5.1: 1D LQR experiment: performance (left, $\gamma = 0.9$) and batch size (right) per episode of SPG, SSPG and AdaBatch, with 95% Student's t-confidence intervals over 5 runs.

make SPG very slow and data-inefficient, preventing application to large-scale practical problems. However, numerical simulations improve the understanding of the algorithms and further support the theoretical claims.

One-dimensional LQR. Figure 5.1 shows the learning behavior of SPG on the one-dimensional *Linear Quadratic Regulator* (LQR) task (Section C.1) with horizon $H = 10$, discount factor $\gamma = 0.9$, $\mathcal{S} = \mathcal{A} = [-1, 1]$, $A = B = C = D = 1$, and a Gaussian policy with mean linear in the state, fixed standard deviation $\sigma = 1$, and initial mean parameter $\theta = 0$. On the left, we report the average return *per trajectory*. Even on this simple task, SPG requires very large batch sizes (order of 10^4) and is very slow to converge. The algorithm by Papini et al. (2017) (AdaBatch in the figure) is faster, likely because it uses a Bernstein inequality. With Gaussian confidence intervals, SSPG shows a significant improvement and is able to converge within a reasonable time.

On the right of Figure 5.1, we can see how the batch size changes during the learning process. Both SPG and AdaBatch increase it in a steady manner as they approach convergence. Intuitively, the gradient magnitude becomes small w.r.t. its variance and more samples are needed to reliably estimate the improvement direction. The batch size of SSPG is more oscillating since it is based on the empirical variance of the gradient estimate. However, it consistently employs smaller batch sizes than SPG in the early phases of the learning process, which allows to perform more policy updates and converge faster.

The fluctuations that can be observed in the empirical performance (Figure 5.1, left) are due to the stochasticity of the interaction (random initialization and policy variance) and not to oscillations of the expected performance. To verify this, we

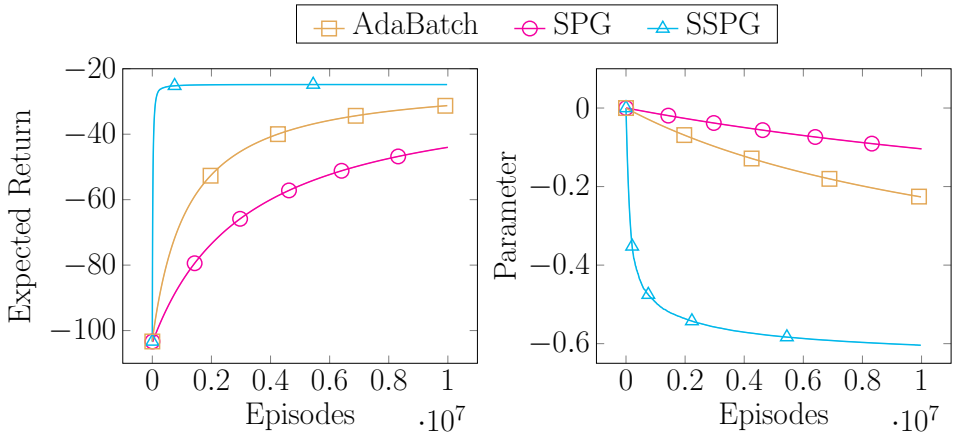


Figure 5.2: 1D LQR experiment: theoretical performance and policy parameter per episode of SPG, SSPG and AdaBatch, with 95% Student's t-confidence intervals over 5 runs (too tight to be appreciated).

trace the policy parameter during the learning process and compute the theoretical performance with dynamic programming. Both are visible in Figure 5.2 and are indeed monotonic curves.[5] The theoretically optimal parameter is $\theta^* = -0.58$, which is close to the value learned by SSPG. Note also how the variability in the learning behavior between different runs is actually negligible.

Three-dimensional LQR. The next experiment is on a LQR problem with three-dimensional state and scalar action. The state space is $[-1, 1]^3$, the action space is $[-4, 4]$. The system matrices are as follows:

$$A = \begin{bmatrix} 1 & 0.1 & 0 \\ 0 & 1 & -0.05 \\ 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0.5 \\ 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0.7 & 0.5 & 0 \end{bmatrix}, \quad D = 0.1.$$

We use a shallow Gaussian policy with three mean parameters initialized at zero and a fixed standard deviation $\sigma = 1$. Performance oscillations can be observed by running G(PO)MDP with a large step size ($\alpha = 1.3$). SSPG is able to prevent these oscillations. However, G(PO)MDP with a constant step ($\alpha = 0.13$) and batch size ($N = 100$) also displays monotonic improvement while converging must faster than SSPG. This gap between theory and practice is due to the worst-case assumptions embedded in the choice of meta-parameters performed by the safe algorithm. We discuss it further in the next section.

[5] The scale of the theoretical performance is different because it considers an infinite horizon and unclipped states and actions.

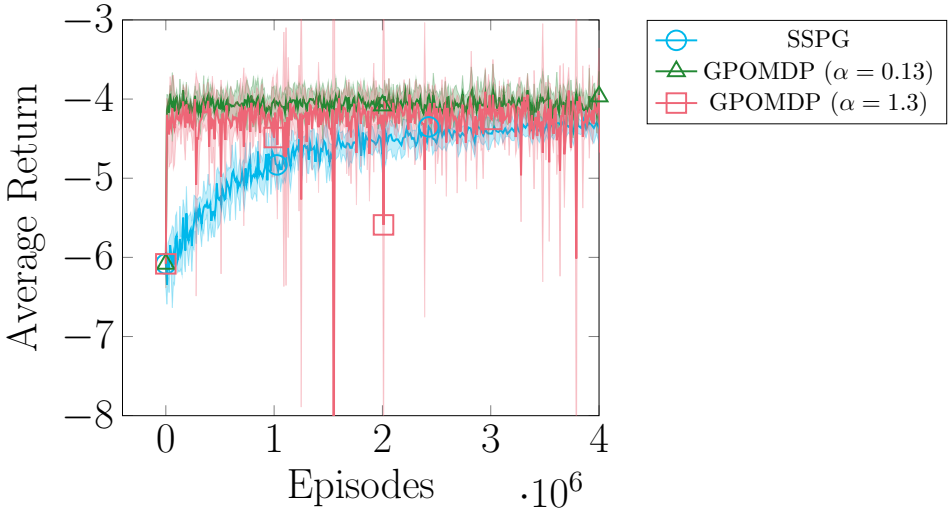


FIGURE 5.3: 3D LQR experiment: performance of SSPG and GPOMDP (with different step sizes); 95% Student's t-confidence intervals over 5 runs.

5.10 On the Theory-Practice Gap

The gap between theory and practice highlighted by the numerical simulations of Section 5.9 could seem disappointing given our original concrete motivations on safe learning in the real world. However, monotonic improvement guarantees should be taken more as a theoretical certificate of the appropriateness of policy gradient methods rather than as a practical tool, much like convergence guarantees in non-convex optimization. Indeed, the conservative step size used by SPG is of the same kind (inverse gradient Lipschitz constant) of the ones used to prove the convergence of stochastic gradient descent and policy gradient methods themselves (see Chapter 7). We want to stress an important difference between our approach and the related one by Schulman et al. (2015a). The derivation of TRPO starts from very general theoretical guarantees that cannot be implemented for policy gradient methods in continuous MDPs, then applies a series of approximations to produce a practical algorithm. The latter is very effective, but loses the guarantees. Instead, our work lies where theory and practice meet, by providing an algorithm that can actually be implemented without compromises and guarantees monotonic improvement (with high probability). To close the gap, we prefer to relax the safety requirements rather than introduce approximations to the algorithm itself. We think this is more coherent with the Seldonian view of safe RL, since we can let the user decide the amount of relaxation. We will explore a possible way to do so in Chapter 8. A similar philosophy guides the design of a practical semi-offline policy optimization algorithm in the following chapter.

Conservative Off-Policy Optimization 6

In Chapter 5, we have remarked the fact that data (trajectories) are a precious and hard-to-obtain resource for RL algorithms, more so if they depend on direct interaction with a physical system. So, we would be tempted to re-use experience data many times to boost efficiency. In this chapter, we show the limitations of this approach and propose conservative policy optimization algorithms that rely on the already available data as much as possible, but not more.

Our approach is based on *importance sampling*, a technique originally developed for off-distribution Monte Carlo estimation (Owen, 2013). We begin this chapter by reviewing the basics of this technique in Section 6.1, where we provide an important characterization of the variance of importance-weighted estimators. In Section 6.2, we explain how importance sampling can be used to optimize a function from off-distribution data, show the risks due to the variance of importance weights, and propose a way to mitigate this risk. We apply this approach to off-policy policy optimization in Section 6.3. Our *Policy Optimization via Importance Sampling* (POIS) algorithm alternates between online interaction and conservative offline policy improvement. We introduce two main flavors of POIS: one for action-based exploration, generalizing REINFORCE, and one for parameter-based exploration, generalizing PGPE. We also provide variants employing *multiple importance sampling*, which allow to re-use all past data instead of only the most recent ones, and per-decision importance sampling (Precup et al., 2000). In Section 6.4, we empirically evaluate the different versions of POIS on continuous-control benchmark tasks, obtaining results comparable to those of state-of-the-art policy optimization algorithms on most problems.

This chapter is based on work done with Metelli et al. (2018, 2020b).

6.1 Off-Distribution Estimation

Consider the general problem of estimating the expected value of a deterministic function f of random variable x taking values in $\mathcal{X}$ under a target distribution P, having at our disposal datasets of samples collected with J behavioral distributions $Q_{1:J} = \{Q_j\}_{j=1}^{J}$.

6.1.1 Importance Sampling

When the available samples are drawn from a single distribution Q, i.e., $J = 1$, the *Importance Sampling* (IS) estimator (Cochran, 2007; Owen, 2013) corrects the distribution with the importance weight (or Radon–Nikodym derivative, or likelihood ratio) defined as $w_{P/Q}(x) = \frac{p(x)}{q(x)}$, leading to the estimator:

$$\widehat{\mu}_{P/Q} = \frac{1}{N}\sum_{i=1}^{N}\frac{p(x_i)}{q(x_i)}f(x_i) = \frac{1}{N}\sum_{i=1}^{N} w_{P/Q}(x_i)f(x_i), \tag{6.1}$$

where $\mathbf{x} = (x_1, x_2, \dots, x_N)^T$ are sampled from Q independently and we assume $q(x) > 0$ whenever $f(x)p(x) \neq 0$. This estimator is unbiased, i.e., $\mathbb{E}_{\mathbf{x}\sim Q}[\widehat{\mu}_{P/Q}] = \mathbb{E}_{x\sim P}[f(x)]$, but it may exhibit an undesirable behavior due to the variability of the importance weights, showing, in some cases, infinite variance. Intuitively, the magnitude of the importance weights provides an indication of how much the probability measures P and Q are dissimilar. This notion can be formalized by the Rényi divergence (Rényi et al., 1961; van Erven and Harremoës, 2014), an information-theoretic dissimilarity index between probability measures.

Let P and Q be two probability measures on a measurable space $(\mathcal{X}, \mathcal{F})$ such that $P \ll Q$ (P is absolutely continuous w.r.t. Q) and Q is σ-finite. Let P and Q admit p and q as Lebesgue probability density functions (p.d.f.), respectively. The α-Rényi divergence between P and Q is defined as:

$$D_\alpha(P\|Q) = \frac{1}{\alpha-1}\log\int_{\mathcal{X}}\left(\frac{\mathrm{d}P}{\mathrm{d}Q}\right)^{\alpha}\mathrm{d}Q = \frac{1}{\alpha-1}\log\int_{\mathcal{X}} q(x)\left(\frac{p(x)}{q(x)}\right)^{\alpha}\mathrm{d}x, \tag{6.2}$$

where $\mathrm{d}P/\mathrm{d}Q$ is the Radon–Nikodym derivative of P w.r.t. Q and $\alpha \in [0, \infty]$. Some remarkable cases, defined as limits, are: $\alpha = 1$ when $D_1(P\|Q) = D_{\mathrm{KL}}(P\|Q)$ and $\alpha = \infty$ yielding $D_\infty(P\|Q) = \log \operatorname{ess\,sup}_{\mathcal{X}}\left\{\frac{\mathrm{d}P}{\mathrm{d}Q}\right\}$.[1] Importing the notation from Cortes et al. (2010), we denote the exponentiated α-Rényi divergence as $d_\alpha(P\|Q) = \exp\{D_\alpha(P\|Q)\}$. With little abuse of notation, we will replace $D_\alpha(P\|Q)$ with $D_\alpha(p\|q)$ whenever possible within the context. When P and Q are (multivariate) Gaussian distributions, i.e., $P \sim \mathcal{N}(\boldsymbol{\mu}_P, \boldsymbol{\Sigma}_P)$ and $Q \sim \mathcal{N}(\boldsymbol{\mu}_Q, \boldsymbol{\Sigma}_Q)$, the Rényi divergence admits a closed-form for $\alpha \in [0, \infty)$ (Burbea, 1984):

$$D_\alpha(P\|Q) = \frac{\alpha}{2}(\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q)^T\boldsymbol{\Sigma}_\alpha^{-1}(\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q) - \frac{1}{2(\alpha-1)}\log\frac{\det(\boldsymbol{\Sigma}_\alpha)}{\det(\boldsymbol{\Sigma}_P)^{1-\alpha}\det(\boldsymbol{\Sigma}_Q)^{\alpha}}, \tag{6.3}$$

[1] ess sup is the essential supremum of a measurable function f, i.e., the smallest M such that $f(x) \leqslant M$ *almost everywhere*.

where $\boldsymbol{\Sigma}_\alpha = \alpha\boldsymbol{\Sigma}_Q + (1-\alpha)\boldsymbol{\Sigma}_P$ under the assumption that $\boldsymbol{\Sigma}_\alpha$ is positive-definite. The Rényi divergence can be computed in closed form for several other widely used distributions (Gil et al., 2013).

The Rényi divergence provides a convenient expression for the moments of the importance weights: $\mathbb{E}_{x\sim Q}\left[w_{P/Q}(x)^\alpha\right] = d_\alpha(P\|Q)^{\alpha-1}$. Moreover, we can relate the Rényi divergence with the variance and the essential supremum of the importance weights (Cortes et al., 2010):

$$\underset{x\sim Q}{\mathbb{V}\mathrm{ar}}\left[w_{P/Q}(x)\right] = d_2(P\|Q) - 1, \tag{6.4}$$

$$\operatorname*{ess\,sup}_{x\sim Q}\left\{w_{P/Q}(x)\right\} = d_\infty(P\|Q). \tag{6.5}$$

6.1.2 Self-Normalized Importance Sampling

A commonly used approach to mitigate the variance problem of the IS estimator, is to resort to the *Self Normalized importance sampling* (SN) estimator (Cochran, 2007):

$$\widetilde{\mu}_{P/Q} = \frac{\sum_{i=1}^N w_{P/Q}(x_i)f(x_i)}{\sum_{i=1}^N w_{P/Q}(x_i)} = \sum_{i=1}^N \widetilde{w}_{P/Q}(x_i)f(x_i), \tag{6.6}$$

where $\widetilde{w}_{P/Q}(x) = w_{P/Q}(x)/\sum_{i=1}^N w_{P/Q}(x_i)$ is the self-normalized importance weight. Differently from $\widehat{\mu}_{P/Q}$, $\widetilde{\mu}_{P/Q}$ is biased but consistent (Owen, 2013) and it typically displays a more desirable behavior because of its smaller variance.[2] The problem of assessing the quality of the SN estimator has been extensively studied by the simulation community, producing several diagnostic indexes to detect when the weights might display problematic behavior (Owen, 2013). The *effective sample size* (ESS) was introduced by Kong (1992) as the number of samples drawn from P so that the variance of the Monte Carlo estimator $\widetilde{\mu}_{P/P}$ is approximately equal to the variance of the SN estimator $\widetilde{\mu}_{P/Q}$ computed with N samples. Here we report the original definition and its most common estimate:

$$\begin{aligned} \mathrm{ESS}(P\|Q) &= \frac{N}{\mathbb{V}\mathrm{ar}_{x\sim Q}\left[w_{P/Q}(x)\right] + 1} = \frac{N}{d_2(P\|Q)}, \\ \widehat{\mathrm{ESS}}(P\|Q) &= \frac{1}{\sum_{i=1}^N \widetilde{w}_{P/Q}(x_i)^2}. \end{aligned} \tag{6.7}$$

The ESS has an interesting interpretation: if $d_2(P\|Q) = 1$, i.e., $P = Q$ almost everywhere, then ESS $= N$ since we are performing Monte Carlo estimation. Otherwise, the ESS decreases as the dissimilarity between the two distributions increases. In the literature, other ESS-like diagnostics have been proposed that also account for the nature of f (Martino et al., 2017).

[2] Note that $|\widetilde{\mu}_{P/Q}| \leqslant \|f\|_\infty$. Therefore, its variance is always finite.

6.1.3 Multiple Importance Sampling

The IS estimator can be extended to the case in which we have samples collected with multiple behavioral distributions Q_j, i.e., when $J > 1$. In *Multiple Importance Sampling* (MIS) frameworks (Veach and Guibas, 1995; Owen, 2013) we have a set of $J \geqslant 1$ behavioral distributions $Q_{1:J} = \{Q_j\}_{j=1}^J$ and a dataset $\mathbf{x}_j = (x_{1j}, x_{2j}, ..., x_{N_j j})$ of N_j samples collected independently with Q_j, $j = 1, 2, ..., J$. We denote with $N = \sum_{j=1}^K N_j$ the total number of samples. The resulting estimator is given by:

$$\widehat{\mu}^{\beta}_{P/Q_{1:J}} = \sum_{j=1}^{J} \frac{1}{N_j} \sum_{i=1}^{N_j} \beta_j(x_{ij}) \frac{p(x_{ij})}{q_j(x_{ij})} f(x_{ij}) = \sum_{j=1}^{J} \frac{1}{N_j} \sum_{i=1}^{N_j} \beta_j(x_{ij}) w_{P/Q_j} f(x_{ij}), \quad (6.8)$$

where we assume that $q_j(x) > 0$ whenever $\beta_j(x)p(x)f(x) = 0$ and $\beta_j(x)$ is a *partition of the unity*, i.e., a collection of weight functions for which $\beta_j(x) \geqslant 0$ for all $j = 1, 2, ..., J$ and $\sum_{j=1}^J \beta_j(x) = 1$ for all $x \in \mathcal{X}$. Several choices for the coefficients β_j (Owen, 2013) are possible. A straightforward, but inefficient, choice is to select $\beta_j(x) = N_j/N$, so as to give equal importance to all the samples. Among all the possible choices for β_j (e.g., cutoff, maximum, power heuristics, see Owen, 2013), the most studied, thanks to its desirable theoretical properties, is the *Balance Heuristic* (BH) (Veach and Guibas, 1995), defined as follows:

$$\beta_j^{\mathrm{BH}}(x) = \frac{N_j q_j(x)}{\sum_{k=1}^J N_k q_k(x)}. \quad (6.9)$$

This particular choice has the advantage of canceling out the q_j in the estimator. In this way, the weight of a given sample x_{ij} does not depend on which component of the mixture it comes from. The resulting estimator has the following form:

$$\widehat{\mu}^{\mathrm{BH}}_{P/Q_{1:J}} = \frac{1}{N} \sum_{j=1}^{J} \sum_{i=1}^{N_j} \frac{p(x_{ij})}{\sum_{k=1}^J \frac{N_k}{N} q_k(x_{ij})} f(x_{ij}) = \frac{1}{N} \sum_{j=1}^{J} \sum_{i=1}^{N_j} w^{\mathrm{BH}}_{P/Q_{1:J}}(x_{ij}) f(x_{ij}), \quad (6.10)$$

which can be interpreted as an importance sampling estimator using the mixture of behavioral distributions with mixture weights $\frac{N_k}{N}$, i.e., $\Phi = \sum_{k=1}^K \frac{N_k}{N} Q_k$. Furthermore, this choice of coefficient functions is nearly optimal (Veach and Guibas, 1995, Theorem 1) in terms of variance of the estimator $\widehat{\mu}_{P/Q_{1:J}}$. Although the variance problem is less crucial in the MIS, compared to the IS case, it is possible to combine the MIS estimator with the self-normalization technique. This can be done in two ways: by normalizing the weights separately for each behavioral distribution:

$$\widetilde{\mu}^{\mathrm{BH}}_{P/Q_{1:J}} = \frac{1}{J} \sum_{j=1}^{J} \sum_{i=1}^{N_j} \frac{w^{\mathrm{BH}}_{P/Q_{1:J}}(x_{ij})}{\sum_{k=1}^{N_j} w^{\mathrm{BH}}_{P/Q_{1:J}}(x_{kj})} f(x_{ij}), \quad (6.11)$$

or by normalizing each weight over all available samples:

$$\widetilde{\widetilde{\mu}}^{\mathrm{BH}}_{P/Q_{1:J}} = \sum_{j=1}^{J} \sum_{i=1}^{N_j} \frac{w^{\mathrm{BH}}_{P/Q_{1:J}}(x_{ij})}{\sum_{h=1}^{J} \sum_{k=1}^{N_h} w^{\mathrm{BH}}_{P/Q_{1:J}}(x_{kh})} f(x_{ij}). \quad (6.12)$$

Both reduce to the classic SN when $J = 1$. However, the first normalization at Equation (6.11) degenerates under unit batch sizes, setting all the normalized weights to one. For this reason, we will adopt the second version (6.12) in our experiments.

6.2 Off-Distribution Optimization

We now aim to determine the target distribution P that maximizes $\mathbb{E}_{x\sim P}[f(x)]$ by having datasets of samples collected with J behavioral distributions $Q_{1:J} = \{Q_j\}_{j=1}^{J}$. In this section, we analyze the problem of defining an objective function suitable for this purpose.

The naïve approach would be to directly optimize the estimator $\widehat{\mu}^{\beta}_{P/Q_{1:J}}$ with the data sampled from $Q_1, \dots, Q_J$. This approach has a fundamental problem (even when using the BH). As shown in Section 6.1.1, the IS estimate is less reliable (i.e., displays a larger variance) for target distributions very different from the behavioral one. With enough freedom in choosing P, the optimal solution would assign as much probability mass as possible to the maximum value among $f(x_i)$. Since the IS estimator is clearly unreliable for such an extreme distribution, this kind of optimization is ill-informed and overconfident. For this reason, we adopt a conservative approach and we decide to optimize a statistical *lower bound* of the expected value $\mathbb{E}_{x\sim P}[f(x)]$ which holds with high confidence.

6.2.1 Variance of importance-sampling estimators

We start by analyzing the behavior of the IS estimator, i.e., $J = 1$ and we provide the following result that bounds the variance of $\widehat{\mu}_{P/Q}$ in terms of the Rényi divergence.

Lemma 6.1 *Let P and Q be two probability measures on the measurable space $(\mathcal{X}, \mathcal{F})$ such that $P \ll Q$. Let $\alpha \in [1, \infty]$, $\mathbf{x} = (x_1, x_2, \dots, x_N)^T$ be i.i.d. random variables sampled from Q and $f : \mathcal{X} \to \mathbb{R}$ be a function with bounded $2\alpha/(\alpha-1)$-moment under Q ($\|f\|_{Q,\frac{2\alpha}{\alpha-1}} < \infty$). Then, for any $N > 0$, the variance of the IS estimator $\widehat{\mu}_{P/Q}$ can be upper bounded as:*

$$\underset{\mathbf{x}\sim Q}{\mathbb{V}ar}\left[\widehat{\mu}_{P/Q}\right] \leqslant \frac{1}{N}\|f\|^2_{Q,\frac{2\alpha}{\alpha-1}} d_{2\alpha}\left(P\|Q\right)^{2-\frac{1}{\alpha}}, \tag{6.13}$$

where we used the abbreviation $\mathbf{x} \sim Q$ for denoting $x_i \sim Q$ for all $i = 1, 2, ..., N$ all independent.

Proof

$$\begin{aligned}
\underset{\mathbf{x}\sim Q}{\mathbb{V}\mathrm{ar}}\left[\widehat{\mu}_{P/Q}\right] &= \frac{1}{N}\underset{x_1\sim Q}{\mathbb{V}\mathrm{ar}}\left[\frac{p(x_1)}{q(x_1)}f(x_1)\right] && (6.14)\\
&\leqslant \frac{1}{N}\underset{x_1\sim Q}{\mathbb{E}}\left[\left(\frac{p(x_1)}{q(x_1)}f(x_1)\right)^2\right] && (6.15)\\
&\leqslant \frac{1}{N}\underset{x_1\sim Q}{\mathbb{E}}\left[\left|\frac{p(x_1)}{q(x_1)}\right|^{2\alpha}\right]^{\frac{1}{\alpha}}\underset{x_1\sim Q}{\mathbb{E}}\left[|f(x_1)|^{\frac{2\alpha}{\alpha-1}}\right]^{\frac{\alpha-1}{\alpha}} && (6.16)\\
&= \frac{1}{N}\|f\|^2_{Q,\frac{2\alpha}{\alpha-1}} d_{2\alpha}\left(P\|Q\right)^{2-\frac{1}{\alpha}},
\end{aligned}$$

where the line (6.14) follows form the fact that the random variables x_i are i.i.d., line (6.15) follows from bounding the variance with the second moment, and line (6.16) is derived by applying Hölder's inequality with $p = \alpha$ and $q = \frac{\alpha}{\alpha-1}$. Finally, we exploit the definition of d_α and $\|\cdot\|_{Q,p}$. ■

A useful special case (first derived by Metelli et al., 2018), can be recovered by setting $\alpha = 1$ under the condition that $\|f\|_\infty < +\infty$:

$$\underset{\mathbf{x}\sim Q}{\mathbb{V}\mathrm{ar}}\left[\widehat{\mu}_{P/Q}\right] \leqslant \frac{1}{N}\|f\|^2_\infty d_2\left(P\|Q\right) \leqslant \frac{\|f\|^2_\infty}{\mathrm{ESS}(P\|Q)}. \tag{6.17}$$

When $P = Q$ almost everywhere, we get $\mathbb{V}\mathrm{ar}_{\mathbf{x}\sim Q}\left[\widehat{\mu}_{Q/Q}\right] \leqslant \frac{1}{N}\|f\|^2_\infty$, a well-known upper bound to the variance of the Monte Carlo estimator. Otherwise, the variance scales with ESS instead of N, further justifying the definition of the ESS. While $\widehat{\mu}_{P/Q}$ can have an unbounded variance even if f is bounded, the SN estimator $\widetilde{\mu}_{P/Q}$ is always bounded by $\|f\|_\infty$ and therefore it always has finite variance. Since the normalization term makes all the samples $\widetilde{w}_{P/Q}(x_i)f(x_i)$ interdependent, an exact analysis of its bias and variance is more challenging. Several works adopted approximate methods for providing an expression for its variance (Hesterberg, 1988). See Metelli et al. (2018, Appendix D) for an analysis of bias and variance that employs the concept of Rényi divergence.

A similar bound can be derived for the variance of the MIS estimator:

Lemma 6.2 *Let P and $\{Q_j\}_{j=1}^J$ be probability measures on the measurable space $(\mathcal{X},\mathcal{F})$ such that $P \ll Q_j$ for $j = 1,\dots,J$. Let $\mathbf{x}_j = (x_{1j}, x_{2j},\dots,x_{N_j\,j})^T$ be i.i.d. random variables sampled from Q_j for $j = 1,\dots,J$. Let $\mathbf{x} = (\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_J)^T$ and $\Phi = \sum_{k=1}^J \frac{N_k}{N}Q_k$ be a finite mixture. Let $\alpha \in [1,\infty]$ and $f : \mathcal{X} \to \mathbb{R}$ be a function with bounded $\frac{2\alpha}{\alpha-1}$-moment under Φ ($\|f\|_{\Phi,\frac{2\alpha}{\alpha-1}} < \infty$). Then, the variance of the multiple importance sampling estimator can be upper bounded as:*

$$\underset{\mathbf{x}\sim Q_{1:J}}{\mathbb{V}ar}\left[\widehat{\mu}^{\mathrm{BH}}_{P/Q_{1:J}}\right] \leqslant \frac{1}{N}\|f\|^2_{\Phi,\frac{2\alpha}{\alpha-1}} d_{2\alpha}\left(P\|\Phi\right)^{2-\frac{1}{\alpha}}, \tag{6.18}$$

where we used the abbreviation $\mathbf{x} \sim Q_{1:J}$ for denoting $\mathbf{x}_j \sim Q_j$ for all $j = 1,2,...,J$ all independent.

Proof Consider the following derivation:

$$\begin{aligned}
\underset{\mathbf{x}\sim Q_{1:J}}{\mathbb{V}\mathrm{ar}}\left[\widehat{\mu}^{\mathrm{BH}}_{P/Q_{1:J}}\right] &= \underset{\mathbf{x}\sim Q_{1:J}}{\mathbb{V}\mathrm{ar}}\left[\frac{1}{N}\sum_{j=1}^{J}\sum_{i=1}^{N_j}\frac{p(x_{ij})}{\sum_{k=1}^{K}\frac{N_k}{N}q_k(x_{ij})}f(x_{ij})\right] \\
&= \frac{1}{N^2}\sum_{j=1}^{J}\sum_{i=1}^{N_j}\underset{x_{ij}\sim Q_j}{\mathbb{V}\mathrm{ar}}\left[\frac{p(x_{ij})}{\sum_{k=1}^{J}\frac{N_k}{N}q_k(x_{ij})}f(x_{ij})\right] && (6.19)\\
&\leqslant \frac{1}{N^2}\sum_{j=1}^{J}\sum_{i=1}^{N_j}\underset{x_{ij}\sim Q_j}{\mathbb{E}}\left[\left(\frac{p(x_{ij})}{\sum_{k=1}^{J}\frac{N_k}{N}q_k(x_{ij})}f(x_{ij})\right)^2\right] && (6.20)\\
&= \frac{1}{N}\underset{x\sim\Phi}{\mathbb{E}}\left[\left(\frac{p(x)}{\sum_{k=1}^{J}\frac{N_k}{N}q_k(x)}f(x)\right)^2\right] && (6.21)\\
&\leqslant \frac{1}{N}\underset{x\sim\Phi}{\mathbb{E}}\left[\left|\frac{p(x)}{\sum_{k=1}^{J}\frac{N_k}{N}q_k(x)}\right|^{2\alpha}\right]^{\frac{1}{\alpha}}\underset{x\sim\Phi}{\mathbb{E}}\left[|f(x)|^{\frac{2\alpha}{\alpha-1}}\right]^{\frac{\alpha-1}{\alpha}} && (6.22)\\
&= \frac{1}{N}\|f\|^2_{\Phi,\frac{2\alpha}{\alpha-1}}d_{2\alpha}\left(P\|\Phi\right)^{2-\frac{1}{\alpha}},
\end{aligned}$$

where line (6.19) follows from the fact that all x_{ij} are i.i.d., line (6.20) derives from bounding the variance with the second moment, line (6.21) is obtained from observing that, for a generic function g we have:

$$\frac{1}{N}\sum_{j=1}^{J}\sum_{i=1}^{N_j}\underset{x_{ij}\sim Q_j}{\mathbb{E}}\left[g(x_{ij})\right] = \sum_{j=1}^{J}\frac{N_j}{N}\underset{x_{1j}\sim Q_j}{\mathbb{E}}\left[g(x_{1j})\right] = \underset{x\sim\Phi}{\mathbb{E}}\left[g(x)\right].$$

Then, line (6.22) is obtained by Hölder's inequality with $p=\alpha$ and $q=\frac{\alpha}{\alpha-1}$. ■

Similarly to the single-IS case, an interesting case is obtained when setting $\alpha=2$ and requiring $\|f\|_\infty < +\infty$:

$$\underset{\mathbf{x}\sim Q_{1:J}}{\mathbb{V}\mathrm{ar}}\left[\widehat{\mu}^{\mathrm{BH}}_{P/Q_{1:J}}\right] \leqslant \frac{1}{N}\|f\|^2_\infty d_2\left(P\|\Phi\right). \tag{6.23}$$

While for Gaussian distributions the Rényi divergence can be computed in closed-form (Equation 6.3), when moving to the MIS case we need to evaluate the d_α between a Gaussian distribution and a mixture of Gaussians, which does not admit a closed form. A straightforward approach consists in exploiting the convexity of d_α w.r.t. to the second argument, when $\alpha \geqslant 1$, to obtain the loose bound:

$$d_\alpha(P\|\Phi) \leqslant \sum_{k=1}^{K}\frac{N_k}{N}d_\alpha(P\|Q_k).$$

However, this bound would be vacuous when at least one of the terms $d_\alpha(P\|Q_k)$ is infinite, while, clearly, the variance of the estimator would be finite as long as at

least one of the terms $d_\alpha(P\|Q_k)$ is finite. This intuition is captured by a tighter bound that resorts to the harmonic mean of the terms $d_\alpha(P\|Q_k)$:

Theorem 6.1 *(Papini et al., 2019a, Theorem 5) Let P and $\{Q_j\}_{j=1}^J$ be probability measures on the measurable space $(\mathcal{X}, \mathcal{F})$ such that $P \ll Q_j$ for $j = 1, \dots, J$. Let $\Phi = \sum_{j=1}^J \zeta_j Q_j$, with $\zeta_j \geqslant 0$ for all $j = 1, 2, ..., J$ and $\sum_{j=1}^J \zeta_j = 1$ be a finite mixture. Then, for any $\alpha \in [1, \infty]$, the exponentiated α-Rényi divergence can be bounded as:*

$$d_\alpha(P\|\Phi) \leqslant \frac{1}{\sum_{j=1}^J \frac{\zeta_j}{d_\alpha(P\|Q_j)}}. \tag{6.24}$$

We just need to set $\zeta_j = \frac{N_j}{N}$ in Theorem 6.1 to obtain the case of our interest. It is worth noting that Theorem 6.1 shows that the bound on the variance of the MIS estimator $\widehat{\mu}_{P/Q_{1:J}}$ with BH is never worse than the bound on the variance of the IS estimator $\widehat{\mu}_{P/Q_{j*}}$ that uses the distribution Q_{j*} which is the closest to P among the $Q_{1:J}$. Indeed, we can easily obtain the following inequality:

$$\frac{d_2(P\|\Phi)}{N} \leqslant \frac{1}{\sum_{j=1}^J \frac{N_j}{d_2(P\|Q_j)}} \leqslant \min_{j \in \{1,\dots,J\}} \frac{d_2(P\|Q_j)}{N_j}.$$

6.2.2 Concentration inequalities

We seek a concentration inequality in order to define a conservative objective for off-distribution optimization. On the one hand, fully empirical concentration inequalities, like Student-T, besides the asymptotic approximation, are not suitable in this case since empirical variance needs to be estimated with importance sampling as well, injecting further uncertainty (Owen, 2013). On the other hand, several distribution-free inequalities, like Hoeffding, require knowing the maximum of the estimator, which might not exist for the IS estimator when $d_\infty(P\|Q) = \infty$. Constraining $d_\infty(P\|Q)$ to be finite often introduces unacceptable limitations. For instance, consider the case of univariate Gaussian distributions of the form $\mathcal{N}(\mu, \sigma^2)$, where the standard deviation σ is one of the parameters that must be learned. The constraint on $d_\infty(P\|Q)$ prevents a step that selects a target variance σ^2 larger than the behavioral one. Even Bernstein inequalities (Bercu et al., 2015), are hardly applicable since, for instance, in the case of univariate Gaussian distributions, the importance weights display a *heavy-tail* behavior.[3] We believe that a reasonable trade-off should require the variance of the importance weights to be finite, which is equivalent to require $d_2(P\|Q) < \infty$, i.e., $\sigma_P < 2\sigma_Q$ for univariate Gaussians. For this reason, we resort to Chebyshev-like inequalities and we propose the following concentration bound derived from Cantelli's inequality (Cantelli, 1929) and customized for the IS estimator:

Theorem 6.2 *Let P and Q be two probability measures on the measurable space $(\mathcal{X}, \mathcal{F})$ such that $P \ll Q$ and $d_2(P\|Q) < \infty$. Let $x_1, x_2, \dots, x_N$ be i.i.d. random*

[3] For a detailed analysis of the properties of the IS estimator for Gaussian distributions refer to (Appendix C Metelli et al., 2018)

variables sampled from Q, and $f : \mathcal{X} \to \mathbb{R}$ be a bounded function ($\|f\|_\infty < \infty$). Then, for any $0 < \delta \leqslant 1$ and $N > 0$ with probability at least $1 - \delta$ it holds that:

$$\mathop{\mathbb{E}}_{x\sim P}[f(x)] \geqslant \underbrace{\frac{1}{N}\sum_{i=1}^{N} w_{P/Q}(x_i)f(x_i)}_{\widehat{\mu}_{P/Q}} - \|f\|_\infty \sqrt{\frac{(1-\delta)d_2(P\|Q)}{\delta N}}. \tag{6.25}$$

Proof We start from Cantelli's inequality (Cantelli, 1929) applied on the random variable $\widehat{\mu}_{P/Q} = \frac{1}{N}\sum_{i=1}^{N} w_{P/Q}(x_i)f(x_i)$:

$$\Pr\left(\widehat{\mu}_{P/Q} - \mathop{\mathbb{E}}_{x\sim P}[f(x)] \geqslant \lambda\right) \leqslant \frac{1}{1 + \frac{\lambda^2}{\mathbb{V}\mathrm{ar}_{\mathbf{x}\sim Q}\left[\widehat{\mu}_{P/Q}\right]}}. \tag{6.26}$$

By renaming $\delta = \frac{1}{1+\frac{\lambda^2}{\mathbb{V}\mathrm{ar}_{\mathbf{x}\sim Q}\left[\widehat{\mu}_{P/Q}\right]}}$ and considering the complementary event, we get that with probability at least $1 - \delta$ we have:

$$\mathop{\mathbb{E}}_{x\sim P}[f(x)] \geqslant \widehat{\mu}_{P/Q} - \sqrt{\frac{1-\delta}{\delta}\mathop{\mathbb{V}\mathrm{ar}}_{\mathbf{x}\sim Q}\left[\widehat{\mu}_{P/Q}\right]}. \tag{6.27}$$

By replacing the variance with the bound in Lemma 6.1 (setting $\alpha = 1$) we get the result. ■

The bound highlights the interesting trade-off between the estimated performance and the uncertainty introduced by changing the distribution. The latter enters in the bound as the 2-Rényi divergence between the target distribution P and the behavioral distribution Q. Intuitively, we should trust the estimator $\widehat{\mu}_{P/Q}$ as long as P is not too far from Q. As similar bound can be obtained for the SN estimator (Metelli et al., 2018, Appendix D).

The same result is also applicable to the multiple importance sampling estimator, just by replacing $\widehat{\mu}_{P/Q}$ with $\widehat{\mu}^{\mathrm{BH}}_{P/Q_{1:J}}$ and using the variance bound from Lemma 6.2.

Corollary 6.3 *Let P and $\{Q_j\}_{j=1}^{J}$ be probability measures on the measurable space $(\mathcal{X}, \mathcal{F})$ such that $P \ll Q_j$ for $j = 1, \dots, J$. Let $x_{1j}, x_{2j}, \dots, x_{N_j j}$ be i.i.d. random variables sampled from Q_j with $j = 1, 2, ..., J$, let $\Phi = \sum_{k=1}^{J} \frac{N_k}{N} Q_k$ be a finite mixture such that $d_2(P\|\Phi) < \infty$ and $f : \mathcal{X} \to \mathbb{R}$ be a bounded function ($\|f\|_\infty < \infty$). Then, for any $0 < \delta \leqslant 1$ and $N > 0$, with probability at least $1 - \delta$, it holds that:*

$$\mathop{\mathbb{E}}_{x\sim P}[f(x)] \geqslant \underbrace{\frac{1}{N}\sum_{j=1}^{J}\sum_{i=1}^{N_j} w^{\mathrm{BH}}_{P/Q_{1:J}}(x_{ij})f(x_{ij})}_{\widehat{\mu}^{\mathrm{BH}}_{P/Q_{1:J}}} - \|f\|_\infty \sqrt{\frac{(1-\delta)d_2(P\|\Phi)}{\delta N}}. \tag{6.28}$$

6.2.3 Maximizing the lower bound

Now consider a class $\{P_{\boldsymbol{\omega}} | \boldsymbol{\omega} \in \Omega\}$ of parametric target distributions, all absolutely continuous w.r.t. Q. We maximize the lower bound on expected performance from Theorem (6.2) w.r.t. parameter vector $\boldsymbol{\omega}$:

$$\max_{\boldsymbol{\omega} \in \Omega} \mathcal{L}_\lambda(\boldsymbol{\omega}; Q) \coloneqq \frac{1}{N} \sum_{i=1}^{N} w_{P_{\boldsymbol{\omega}}/Q}(x_i) f(x_i) - \lambda \sqrt{\frac{d_2(P_{\boldsymbol{\omega}} \| Q)}{N}}, \tag{6.29}$$

where $\lambda > 0$ is a penalization coefficient subsuming $\|f\|_\infty \sqrt{(1-\delta)/\delta}$, which can be directly hand-tuned for specific applications. The first term in (6.29) favors values of $\boldsymbol{\omega}$ that put more weight on high observed payoffs. The second term acts as a penalization for distributions that diverge too much from the behavioral one, from which the observed payoffs have been actually generated. From Theorem 6.2, the solution $\mathcal{L}_\lambda(\boldsymbol{\omega}^*)$ of (6.29) lower-bounds, with high probability, the true expected payoff $\mathbb{E}_{x \sim P_{\boldsymbol{\omega}^*}}[f(x)]$. In this way, we are indeed optimizing expected payoff in a conservative way from off-distribution data. A similar optimization problem can be defined for the MIS setting of Corollary 6.3.

Realistically, we cannot solve (6.29) in closed form. However, we can perform gradient ascent on the surrogate objective $\mathcal{L}_\lambda$. Given the off-distribution nature of the objective, we can perform several gradient updates with the same data. In this resides the promised data efficiency of the approach. In Section 6.3, we specialize this abstract problem to policy optimization. In that case, the behavioral distribution Q corresponds to the last policy used to interact with the environment. In the MIS setting, all past policies can serve as behaviorals.

6.2.4 Importance Sampling and Natural Gradient

There is a tight connection between the geometry induced by the Rényi divergence and the Fisher information metric (Section 3.3.1), as pointed out by Metelli et al. (2020b, Theorem 4.4):

Lemma 6.3 *Let $p_{\boldsymbol{\omega}}$ be a p.d.f. differentiable w.r.t. $\boldsymbol{\omega} \in \Omega$. Then, it holds that, for the Rényi divergence:*

$$D_\alpha(p_{\boldsymbol{\omega}'} \| p_{\boldsymbol{\omega}}) = \frac{\alpha}{2} \left(\boldsymbol{\omega}' - \boldsymbol{\omega}\right)^T F(\boldsymbol{\omega}) \left(\boldsymbol{\omega}' - \boldsymbol{\omega}\right) + o(\|\boldsymbol{\omega}' - \boldsymbol{\omega}\|_2^2),$$

and for the exponentiated Rényi divergence:

$$d_\alpha(p_{\boldsymbol{\omega}'} \| p_{\boldsymbol{\omega}}) = 1 + \frac{\alpha}{2} \left(\boldsymbol{\omega}' - \boldsymbol{\omega}\right)^T F(\boldsymbol{\omega}) \left(\boldsymbol{\omega}' - \boldsymbol{\omega}\right) + o(\|\boldsymbol{\omega}' - \boldsymbol{\omega}\|_2^2).$$

This result provides an approximate expression for the variance of the importance weights:

$$\underset{x \sim p_{\boldsymbol{\omega}}}{\mathbb{V}\mathrm{ar}} \left[w_{\boldsymbol{\omega}'/\boldsymbol{\omega}}(x) \right] = d_2(p_{\boldsymbol{\omega}'} \| p_{\boldsymbol{\omega}}) - 1 \simeq \left(\boldsymbol{\omega}' - \boldsymbol{\omega}\right)^T F(\boldsymbol{\omega}) \left(\boldsymbol{\omega}' - \boldsymbol{\omega}\right), \tag{6.30}$$

which can be used to justify the use of natural gradients in off-distribution optimization. Say we want to find the steepest ascent update for objective $\mathcal{L} : \Omega \to \mathbb{R}$ that keeps the variance of the importance weights under control:

$$\begin{aligned} \max_{\Delta\boldsymbol{\omega}} \quad & \nabla_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega})^T \Delta\boldsymbol{\omega} \\ \text{subject to} \quad & \mathop{\mathbb{V}\text{ar}}_{x \sim p_{\boldsymbol{\omega}}} \left[w_{\boldsymbol{\omega}'/\boldsymbol{\omega}}(x) \right] \leqslant \epsilon^2, \end{aligned}$$

for some small threshold $\epsilon > 0$. By approximating the variance with Equation (6.30) and solving the resulting constrained optimization problem we obtain:

$$\Delta\boldsymbol{\omega} = \frac{\epsilon}{\sqrt{\nabla_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega})^T F(\boldsymbol{\omega})^{-1} \nabla_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega})}} F(\boldsymbol{\omega})^{-1} \nabla_{\boldsymbol{\omega}} \mathcal{L}(\boldsymbol{\omega}), \tag{6.31}$$

which is precisely a natural gradient update (Amari, 1998) with the adaptive step size suggested by Matsubara et al. (2010).

Besides motivating the use of natural gradients in our own work, this result can provide further justification to trust-region methods based on natural gradients, such as the original implementation of TRPO (Schulman et al., 2015a, see Section 3.8). Note also that, from (3.11) and Lemma 6.3, the KL divergence is proportional to the variance of importance weights up to the second order. This perspective can also shed new light on the relationship between TRPO and PPO by reversing the original narrative (Schulman et al., 2017b): PPO directly constrains the variance of importance weights by clipping them, while TRPO does that implicitly with a KL constraint.

6.3 Policy Optimization via Importance Sampling

In this section, we discuss how to customize the conservative off-distribution optimization approach (6.29), described in abstract terms in the previous section, for policy optimization. The data efficiency of the proposed approach becomes crucial here since collecting trajectories can be very expensive. We present POIS (first introduced by Metelli et al., 2018), a model-free actor-only policy search algorithm. POIS comes in two flavors, corresponding to two ways of performing exploration: *Parameter–based POIS* (P-POIS), which adopts the parameter-based exploration framework of PGPE (Section 3.12.2), and *Action–based POIS* (A-POIS), which relies on the standard action-based exploration framework of REINFORCE. We then show how to extend these algorithms to the MIS setting to make an even more efficient use of data. In the following we operate under Assumption 5.1, that is, rewards are uniformly bounded by $R_{\max}$.

6.3.1 Parameter-based POIS

For *Parameter–based POIS* (P-POIS), we consider the framework described in Section 3.12.2 for PGPE (Sehnke et al., 2008). Recall that we have a parametric policy space $\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^d\}$, with $\pi_{\boldsymbol{\theta}}$ not necessarily differentiable nor stochastic. The policy parameters $\boldsymbol{\theta}$ are sampled at the beginning of each episode from a

parametric hyperpolicy $\rho_{\boldsymbol{\nu}}$, selected in a parametric space $\mathcal{P}_{\mathcal{V}} = \{\rho_{\boldsymbol{\nu}} : \boldsymbol{\nu} \in \mathcal{V} \subseteq \mathbb{R}^m\}$. Hyperpolicies must be stochastic and differentiable w.r.t. $\boldsymbol{\nu}$. The goal is to maximize $J(\boldsymbol{\nu})$ as defined in Equation (3.117). In this setting, the distributions Q and P of Section 6.2 correspond to the behavioral $\rho_{\boldsymbol{\nu}}$ and target $\rho_{\boldsymbol{\nu}'}$ hyperpolicies, while f is the trajectory return $R(\tau)$. The importance weights must take into account all sources of randomness, derived from sampling a policy parameter $\boldsymbol{\theta}$ and a trajectory τ (Zhao et al., 2013):

$$w_{\boldsymbol{\nu}'/\boldsymbol{\nu}}(\boldsymbol{\theta}) = \frac{\rho_{\boldsymbol{\nu}'}(\boldsymbol{\theta})p(\tau|\boldsymbol{\theta})}{\rho_{\boldsymbol{\nu}}(\boldsymbol{\theta})p(\tau|\boldsymbol{\theta})} = \frac{\rho_{\boldsymbol{\nu}'}(\boldsymbol{\theta})}{\rho_{\boldsymbol{\nu}}(\boldsymbol{\theta})}.$$

Note that, from Assumption 5.1, it follows that the trajectory return is bounded by $R_{\max}\frac{1-\gamma^H}{1-\gamma}$ if $\gamma < 1$ and $R_{\max}H$ if $\gamma = 1$. We can then rephrase the surrogate objective from (6.29) for P-POIS:

$$\mathcal{L}_{\lambda}^{\text{P-POIS}}(\boldsymbol{\nu}'/\boldsymbol{\nu}) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} w_{\boldsymbol{\nu}'/\boldsymbol{\nu}}(\boldsymbol{\theta}_i)R(\tau_i)}_{\widehat{J}^{\text{P-POIS}}(\boldsymbol{\nu}'/\boldsymbol{\nu})} - \lambda\sqrt{\frac{d_2\left(\rho_{\boldsymbol{\nu}'}\|\rho_{\boldsymbol{\nu}}\right)}{N}}, \tag{6.32}$$

where λ is a regularization parameter.[4] Each trajectory τ_i is obtained by running an episode with base policy $\pi_{\boldsymbol{\theta}_i}$, and the corresponding policy parameters $\boldsymbol{\theta}_i$ are sampled independently from hyperpolicy $\rho_{\boldsymbol{\nu}}$ at the beginning of each episode $i = 1, 2, ..., N$.

In the policy optimization framework, the major advantage of the MIS estimation, over the standard IS, is the higher sample-efficiency. Indeed, using MIS we can reuse the trajectories generated by all past policies $\{\pi_{B_j}\}_{j=1}^{J}$ to estimate the performance of the target policy π_T. Differently, with the IS estimator we just reuse the trajectories generated by a single policy π_B, usually the last one. This advantage comes at the cost of higher computational complexity, as we need to evaluate the density function induced by each policy for all the collected trajectories. Moving to the MIS framework, we need to redefine the importance weight to account for the multiple behavioral hyperpolicies, having hyperparameters $\boldsymbol{\nu}_{1:J} = \{\boldsymbol{\nu}_j\}_{j=1}^{J}$:

$$w_{\boldsymbol{\nu}'/\boldsymbol{\nu}_{1:J}}^{\text{BH}}(\boldsymbol{\theta}) = \frac{\rho_{\boldsymbol{\nu}'}(\boldsymbol{\theta})p(\tau|\boldsymbol{\theta})}{\sum_{k=1}^{J}\frac{N_k}{N}\rho_{\boldsymbol{\nu}_k}(\boldsymbol{\theta})p(\tau|\boldsymbol{\theta})} = \frac{\rho_{\boldsymbol{\nu}'}(\boldsymbol{\theta})}{\sum_{k=1}^{J}\frac{N_k}{N}\rho_{\boldsymbol{\nu}_k}(\boldsymbol{\theta})},$$

where each N_k counts the number of episodes hyperpolicy $\rho_{\boldsymbol{\nu}_k}$ was sampled from. Therefore, by employing Corollary 6.3 and Theorem 6.1, we can formulate the new surrogate objective:

$$\mathcal{L}_{\lambda}^{\text{P-POIS}}(\boldsymbol{\nu}'/\boldsymbol{\nu}_{1:J}) = \underbrace{\frac{1}{N}\sum_{j=1}^{J}\sum_{i=1}^{N_j} w_{\boldsymbol{\nu}'/\boldsymbol{\nu}_{1:J}}^{\text{BH}}(\boldsymbol{\theta}_{ij})R(\tau_{ij})}_{\widehat{J}^{\text{P-POIS}}(\boldsymbol{\nu}'/\boldsymbol{\nu}_{1:J})} - \frac{\lambda}{\sqrt{\sum_{j=1}^{J}\frac{N_j}{d_2\left(\rho_{\boldsymbol{\nu}'}\|\rho_{\boldsymbol{\nu}_j}\right)}}}, \tag{6.33}$$

[4] For Theorem 6.2 to hold, $\lambda = R_{\max}\frac{1-\gamma^H}{1-\gamma}\sqrt{\frac{1-\delta}{\delta}}$ for $\gamma < 1$ and $\lambda = R_{\max}H\sqrt{\frac{1-\delta}{\delta}}$ for $\gamma = 1$.

where each $\boldsymbol{\theta}_{ij}$ is sampled independently from $\rho_{\boldsymbol{\nu}_j}$ and the corresponding trajectory τ_j is obtained by running policy $\pi_{\boldsymbol{\theta}_{ij}}$ in the environment with $i = 1, 2, ..., N_j$ and $j = 1, 2, ..., J$. Note that (6.33) reduces to (6.32) when setting $J = 1$, i.e., when considering a single behavioral hyperpolicy.

To derive a practical algorithm, we use as behavioral hyperpolicies the J most recent hyperpolicies and we denote them with $\boldsymbol{\nu}_{1:J}$. At each epoch $h = 1, 2, ..., M_{\text{on-line}}$, we sample N_J parameters $\{\boldsymbol{\theta}_i^h\}_{i=1}^{N_J}$ independently from $\rho_{\boldsymbol{\nu}_0^h}$. For each of the $\boldsymbol{\theta}_i^h$, we collect a single trajectory τ_i^h by running policy $\pi_{\boldsymbol{\theta}_i^h}$ in the environment and we observe its return $R(\tau_i^h)$. We now employ this return and all the ones previously collected to optimize the objective function $\mathcal{L}_\lambda^{\text{P-POIS}}$ off-line. In particular, for each (offline) iteration $k = 1, 2, ..., M_{\text{off-line}}$, we compute the gradient $\nabla_{\boldsymbol{\nu}_k^h} \mathcal{L}_\lambda^{\text{P-POIS}}(\boldsymbol{\nu}_k^h / \boldsymbol{\nu}_{1:J}^h)$ of the objective function and we determine a step size α_k using a *line search* procedure. We update the hyperpolicy parameters via gradient ascent:

$$\boldsymbol{\nu}_{k+1}^h = \boldsymbol{\nu}_k^h + \alpha_k \nabla_{\boldsymbol{\nu}_k^h} \mathcal{L}_\lambda^{\text{P-POIS}}(\boldsymbol{\nu}_k^h / \boldsymbol{\nu}_{1:J}^h).$$

Finally, when the off-line optimization is performed, we update the set of behavioral hyperpolicies by removing the oldest parametrization $\boldsymbol{\nu}_0^{h-J}$ and inserting the most recent $\boldsymbol{\nu}_0^{h+1}$. Clearly, the removal of the oldest one needs to be performed only if we have performed at least J on-line iterations, i.e., if $h \geqslant J$. Refer to Algorithm 6 for the complete pseudo-code of P-POIS, presented in the more general MIS version.

A natural-gradient variant can be obtained by simply pre-multiplying the gradient of the objective function by the inverse of the FIM:

$$\boldsymbol{\nu}_{k+1}^h = \boldsymbol{\nu}_k^h + \alpha_k F(\boldsymbol{\nu}_k^h)^{-1} \nabla_{\boldsymbol{\nu}_k^h} \mathcal{L}_\lambda^{\text{P-POIS}}(\boldsymbol{\nu}_k^h / \boldsymbol{\nu}_{1:J}^h). \tag{6.34}$$

If we employ Gaussian hyperpolicies, we can leverage the same properties discussed for PGPE (Section 3.12.2) for efficient computation of natural gradients.

6.3.2 Action-based POIS

In A-POIS we search for a policy that maximizes the performance $J(\boldsymbol{\theta})$ within a parametric space $\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^d\}$ of stochastic differentiable policies. In this context, the behavioral (respectively target) distribution Q (resp. P) becomes the distribution over trajectories $p_{\boldsymbol{\theta}}$ (resp. $p_{\boldsymbol{\theta}'}$) induced by the behavioral policy $\pi_{\boldsymbol{\theta}}$ (resp. target policy $\pi_{\boldsymbol{\theta}'}$) and f is again the trajectory return $R(\tau)$. The corresponding importance weight is defined in terms of trajectory density functions, and reduces to a product of policy ratios:

$$w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau) = \frac{p_{\boldsymbol{\theta}'}(\tau)}{p_{\boldsymbol{\theta}}(\tau)} = \prod_{t=0}^{H-1} \frac{\pi_{\boldsymbol{\theta}'}(a_{\tau,t}|s_{\tau,t})}{\pi_{\boldsymbol{\theta}}(a_{\tau,t}|s_{\tau,t})}, \tag{6.35}$$

since the transition probabilities cancel out (cf. Equation 2.44). The Rényi divergence we need is between the distributions over trajectories induced by the policies. The

Algorithm 6 P-POIS (Parameter-based POIS)

Input: J number of behavioral hyperpolicies
N_J number of samples to collect for each hyperpolicy
$M_{\text{on-line}}$ maximum number of on-line iterations
$M_{\text{off-line}}$ maximum number of off-line iterations
$\lambda \geqslant 0$ regularization parameter
initial hyperpolicy parameters $\boldsymbol{\nu}_0$
Initialize the target hyperpolicy: $\boldsymbol{\nu}_0^0 = \boldsymbol{\nu}_0$
Initialize the behavioral hyperpolicy set: $\boldsymbol{\nu}_{1:J}^0 = \{\boldsymbol{\nu}_0^0\}$
for $h = 0, 1, ..., M_{\text{on-line}} - 1$ **do**
  Sample N_J policy parameters $\{\boldsymbol{\theta}_i^h\}_{i=1}^{N_J}$ independently from $\rho_{\boldsymbol{\nu}_0^h}$
  Sample N_J trajectories $\{\tau_i^h\}_{i=1}^{N_J}$ independently with each $\{\pi_{\boldsymbol{\theta}_i^h}\}_{i=1}^{N_J}$
  for $k = 0, 1, ..., M_{\text{off-line}} - 1$ **do**
    Compute gradient $\nabla_{\boldsymbol{\nu}_k^h} \mathcal{L}_\lambda^{\text{P-POIS}}(\boldsymbol{\nu}_k^h / \boldsymbol{\nu}_{1:J}^h)$
    Select the step size α_k^h using line search
    Update the hyperpolicy parameters $\boldsymbol{\nu}_{k+1}^h = \boldsymbol{\nu}_k^h + \alpha_k^h \nabla_{\boldsymbol{\nu}_k^h} \mathcal{L}_\lambda^{\text{P-POIS}}(\boldsymbol{\nu}_k^h / \boldsymbol{\nu}_{1:J}^h)$
  end for
  Update the last behavioral hyperpolicy $\boldsymbol{\nu}_0^{h+1} = \boldsymbol{\nu}_{M_{\text{off-line}}}^h$
  Update the behavioral hyperpolicy set $\boldsymbol{\nu}_{1:J}^{h+1} = \begin{cases} \left(\boldsymbol{\nu}_{1:J}^h \backslash \{\boldsymbol{\nu}_0^{h-J}\}\right) \cup \{\boldsymbol{\nu}_0^{h+1}\} & \text{if } h \geqslant J \\ \boldsymbol{\nu}_{1:J}^h \cup \{\boldsymbol{\nu}_0^{h+1}\} & \text{otherwise} \end{cases}$
end for

surrogate objective for A-POIS is then:

$$\mathcal{L}_\lambda^{\text{A-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta}) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_i) R(\tau_i)}_{\hat{J}^{\text{A-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta})} - \lambda \sqrt{\frac{d_2\left(p_{\boldsymbol{\theta}'} \| p_{\boldsymbol{\theta}}\right)}{N}}, \tag{6.36}$$

Differently from P-POIS, the surrogate objective function cannot be directly optimized via gradient ascent since computing $d_2\left(p_{\boldsymbol{\theta}'} \| p_{\boldsymbol{\theta}}\right)$ requires the approximation of an integral over the trajectory space, even for well-known policy models (like Gaussian policies). Furthermore, for stochastic environments, we need to know the functional form of the transition model p:

$$d_2\left(p_{\boldsymbol{\theta}'} \| p_{\boldsymbol{\theta}}\right) = \int_{\mathcal{T}} p_{\boldsymbol{\theta}}(\tau) \left(\frac{p_{\boldsymbol{\theta}'}(\tau)}{p_{\boldsymbol{\theta}}(\tau)}\right)^2 \mathrm{d}\tau = \int_{\mathcal{T}} p_{\boldsymbol{\theta}}(\tau) \left(\prod_{t=0}^{H-1} \frac{\pi_{\boldsymbol{\theta}'}(a_{\tau,t}|s_{\tau,t})}{\pi_{\boldsymbol{\theta}}(a_{\tau,t}|s_{\tau,t})}\right)^2 \mathrm{d}\tau. \tag{6.37}$$

A possible conservative approach is to upper bound $d_2\left(p_{\boldsymbol{\theta}'} \| p_{\boldsymbol{\theta}}\right)$ with the Rényi divergence between the policies, as justified by the following result (Metelli et al., 2020b, Proposition 5.1):

Lemma 6.4 *Let $p_{\boldsymbol{\theta}}$ and $p_{\boldsymbol{\theta}'}$ be the behavioral and target trajectory probability density functions. If $p_{\boldsymbol{\theta}'} \ll p_{\boldsymbol{\theta}}$ and $H < \infty$, then, for any $\alpha \in [0, \infty]$ it holds that:*

$$d_\alpha\left(p_{\boldsymbol{\theta}'} \| p_{\boldsymbol{\theta}}\right) \leqslant \sup_{s \in \mathcal{S}} \left\{d_\alpha\left(\pi_{\boldsymbol{\theta}'}(\cdot|s) \| \pi_{\boldsymbol{\theta}}(\cdot|s)\right)\right\}^H.$$

This bound, besides being hard to compute due to the presence of the supremum, is extremely conservative since the Rényi divergence is raised to the horizon H. For these reasons, in practice, we resort to Rényi -divergence estimation. In particular, we employ the following:

$$\widehat{d}_2\left(p_{\boldsymbol{\theta}'} \| p_{\boldsymbol{\theta}}\right) = \frac{1}{N} \sum_{i=1}^{N} \prod_{t=0}^{H-1} d_2\left(\pi_{\boldsymbol{\theta}'}(\cdot|s_t^{(i)}) \| \pi_{\boldsymbol{\theta}}(\cdot|s_t^{(i)})\right). \tag{6.38}$$

Refer to Metelli et al. (2020b) for an analysis of the bias and variance of this estimator and a thorough comparison with possible alternatives.

The MIS extension of A-POIS is straightforwardly obtained by considering the set of behavioral policies induced by the corresponding parameters $\boldsymbol{\theta}_{1:J} = \{\boldsymbol{\theta}_j\}_{j=1}^{J}$:

$$w^{\mathrm{BH}}_{\boldsymbol{\theta}'/\boldsymbol{\theta}_{1:J}}(\tau) = \frac{p(\tau|\boldsymbol{\theta}')}{\sum_{k=1}^{J} \frac{N_k}{N} p(\tau|\boldsymbol{\theta}_j)} = \frac{\prod_{t=0}^{H-1} \pi_{\boldsymbol{\theta}'}(a_{\tau,t}|s_{\tau,t})}{\sum_{k=1}^{J} \prod_{t=0}^{H-1} \frac{N_k}{N} \pi_{\boldsymbol{\theta}_j}(a_{\tau,t}|s_{\tau,t})}.$$

From Corollary 6.3 and Theorem 6.1, a valid surrogate objective is then:

$$\mathcal{L}_{\lambda}^{\mathrm{A-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta}_{1:J}) = \underbrace{\frac{1}{N} \sum_{j=1}^{J} \sum_{i=1}^{N_j} w^{\mathrm{BH}}_{\boldsymbol{\theta}'/\boldsymbol{\theta}_{1:J}}(\tau_{ij}) R(\tau_{ij})}_{\widehat{J}^{\mathrm{A-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta}_{1:J})} - \frac{\lambda}{\sqrt{\sum_{j=1}^{J} \frac{N_j}{d_2(p_{\boldsymbol{\theta}'} \| p(\cdot|\boldsymbol{\theta}_j))}}}, \tag{6.39}$$

where each τ_{ij} is obtained by running policy $\pi_{\boldsymbol{\theta}_j}$ in the environment with $i = 1, 2, ..., N_j$ and $j = 1, 2, ..., J$.

The learning process proceeds in a way similar to P-POIS. We consider the J most recent policies $\boldsymbol{\theta}_{1:J}$ as behavioral policies. At each epoch $h = 1, 2, ..., M_{\text{on-line}}$, we collect N_J trajectories $\{\theta_i^h\}_{i=1}^{N_J}$ independently from $\pi_{\boldsymbol{\theta}_0^h}$ and we observe their return $R(\tau_i^h)$. These trajectories are then used to perform off-line optimization of the objective function $\mathcal{L}_{\lambda}^{\mathrm{A-POIS}}$. More specifically, for each (off-line) iteration $k = 1, 2, ..., M_{\text{off-line}}$, we compute the gradient $\nabla_{\boldsymbol{\theta}_k^h} \mathcal{L}_{\lambda}^{\mathrm{A-POIS}}(\boldsymbol{\theta}_k^h/\boldsymbol{\theta}_{1:J}^h)$ of the objective function and we determine a step size using a line search procedure. The policy parametrization is then updated via gradient ascent:

$$\boldsymbol{\theta}_{k+1}^h = \boldsymbol{\theta}_k^h + \alpha_k \nabla_{\boldsymbol{\theta}_k^h} \mathcal{L}_{\lambda}^{\mathrm{A-POIS}}(\boldsymbol{\theta}_k^h/\boldsymbol{\theta}_{1:J}^h).$$

Finally, we update the set of behavioral policies by inserting the new parametrization $\boldsymbol{\theta}_0^{h+1}$ and, if $h \geqslant J$, removing the oldest one $\boldsymbol{\theta}_0^{h-J}$. Refer to Algorithm 7 for the complete pseudo-code of A-POIS.

Designing natural-gradient variant is more challenging in this case, since we would also need to estimate the FIM from off-policy data.

6.3.3 Per–decision action–based POIS

In the previous section, we introduced A-POIS by defining a unique importance weight $w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau)$ for a whole trajectory τ. However, we can refine the estimator

Algorithm 7 A-POIS (Action-based POIS)

Input: J number of behavioral policies
N_J number of samples to collect for each policy
$M_{\text{on-line}}$ maximum number of on-line iterations
$M_{\text{off-line}}$ maximum number of off-line iterations
$\lambda \geqslant 0$ regularization parameter
Initial policy parameters $\boldsymbol{\theta}_0$
Initialize the target policy: $\boldsymbol{\theta}_0^0 = \boldsymbol{\theta}_0$
Initialize the behavioral policy set $\boldsymbol{\theta}_{1:J}^0 = \{\boldsymbol{\theta}_0^0\}$
for $h = 0, 1, ..., M_{\text{on-line}} - 1$ **do**
  Sample N_J trajectories $\{\tau_i^h\}_{i=1}^{N_J}$ independently from $\pi_{\boldsymbol{\theta}_0^h}$
  for $k = 0, 1, ..., M_{\text{off-line}} - 1$ **do**
    Compute gradient $\nabla_{\boldsymbol{\theta}_k^h} \mathcal{L}_\lambda^{\text{A-POIS}}(\boldsymbol{\theta}_k^h / \boldsymbol{\theta}_{1:J}^h)$
    Select the step size α_k^h using line search
    Update the policy parameters $\boldsymbol{\theta}_{k+1}^h = \boldsymbol{\theta}_k^h + \alpha_k \nabla_{\boldsymbol{\theta}_k^h} \mathcal{L}_\lambda^{\text{A-POIS}}(\boldsymbol{\theta}_k^h / \boldsymbol{\theta}_{1:J}^h)$
  end for
  Update the last behavioral policy $\boldsymbol{\theta}_0^{h+1} = \boldsymbol{\theta}_{M_{\text{off-line}}}^h$
  Update the behavioral policy set $\boldsymbol{\theta}_{1:J}^{h+1} = \begin{cases} (\boldsymbol{\theta}_{1:J}^h \backslash \{\boldsymbol{\theta}_0^{h-J}\}) \cup \{\boldsymbol{\theta}_0^{h+1}\} & \text{if } h \geqslant J \\ \boldsymbol{\theta}_{1:J}^h \cup \{\boldsymbol{\theta}_0^{h+1}\} & \text{otherwise} \end{cases}$
end for

$\widehat{J}^{\text{A-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta})$ by observing that, given a time step $t \in \{0, 1, ..., H-1\}$, the corresponding reward $R(s_{\tau,t}, a_{\tau,t})$ does not depend on actions and states visited after t. Thus, to reweigh the reward at time t, we can limit the importance weight to consider the products of policy ratios *up to* t. This is the rationale behind the introduction of *Per-Decision Importance Sampling* (PDIS) (Precup et al., 2000). In the action-based exploration framework, per–decision importance weights can be defined, for each time step t, as:

$$w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t}) = \frac{p_{\boldsymbol{\theta}'}(\tau_{0:t})}{p_{\boldsymbol{\theta}}(\tau_{0:t})} = \prod_{k=0}^{t} \frac{\pi_{\boldsymbol{\theta}'}(a_k|s_k)}{\pi_{\boldsymbol{\theta}}(a_k|s_k)}, \tag{6.40}$$

for $t = 0, 1, \ldots, H-1$, where $\tau_{0:t}$ denotes the trajectory prefix up to time t (cf. Section 2.5.1).

By using the weights defined in Equation (6.40), we can provide the following estimator for the expected return:

$$\widehat{J}^{\text{D-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H-1} \gamma^t w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t}^{(i)}) R(s_t^{(i)}, a_t^{(i)}). \tag{6.41}$$

Per-decision IS preserves the unbiasedness of the estimator, indeed:

$$\mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} [w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t}) R(s_t, a_t)] = \mathop{\mathbb{E}}_{\tau \sim p_{\boldsymbol{\theta}}} [w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau) R(s_t, a_t)], \tag{6.42}$$

for all $t \in \{0, 1, ..., H-1\}$. It is worth noting that the importance weight employed in A-POIS is obtained by setting $t = H - 1$ in Equation (6.40). Intuitively, by

considering a product made up of fewer policy ratios, we might gain an advantage in terms of injected uncertainty. Unfortunately, the variance of the PDIS estimator is *not* always smaller than the variance of the corresponding "naive" IS estimator (see Metelli et al., 2020b, Fact 5.1, for a counterexample). We may still have a variance-reduction effect in practice, but this is task-dependent. See Metelli et al. (2020b); Rowland et al. (2020); Liu et al. (2019b) for further discussions on the topic.

To obtain a per-decision version of A-POIS, we first need a bound on the variance of the PDIS estimator:

Lemma 6.5 *Let $\widehat{J}^{\mathrm{D-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta})$ be the PDIS estimator of the expected return $J(\boldsymbol{\theta}')$ computed with N i.i.d. trajectories $\tau_1, \tau_2, \dots, \tau_N$ collected running $\pi_{\boldsymbol{\theta}}$, as defined in Equation (6.41). If $p_{\boldsymbol{\theta}'}(\tau_{0:t}) \ll p_{\boldsymbol{\theta}}(\tau_{0:t})$ for all $t = 0, 1, ..., H-1$, then the variance of $\widehat{J}^{\mathrm{D-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta})$ can be upper bounded as:*

$$\mathbb{V}ar\left[\widehat{J}^{\mathrm{D-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta})\right] \leqslant \frac{R_{\max}^2}{N} \sum_{t=0}^{H-1} c_t d_2\left(p_{\boldsymbol{\theta}'}(\tau_{0:t}) \| p_{\boldsymbol{\theta}}(\tau_{0:t})\right), \tag{6.43}$$

where c_t is defined as:

$$c_t = \begin{cases} \dfrac{\gamma^t\left(\gamma^t + \gamma^{t+1} - 2\gamma^H\right)}{1-\gamma} & \text{if } \gamma < 1 \\ 2H - 2t - 1 & \text{if } \gamma = 1 \end{cases}.$$

Proof

$$\mathbb{V}\mathrm{ar}\left[\widehat{J}^{\mathrm{D-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta})\right] = \frac{1}{N} \mathop{\mathbb{V}\mathrm{ar}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{H-1} \gamma^t w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t}) R(s_t, a_t)\right] \tag{6.44}$$

$$\leqslant \frac{1}{N} \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\left(\sum_{t=0}^{H-1} \gamma^t w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t}) R(s_t, a_t)\right)^2\right] \tag{6.45}$$

$$\leqslant \frac{R_{\max}^2}{N} \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\left(\sum_{t=0}^{H-1} \gamma^t w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t})\right)^2\right] \tag{6.46}$$

$$= \frac{R_{\max}^2}{N} \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{H-1} \gamma^{2t} w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t})^2 + 2\sum_{t=0}^{H-2}\sum_{k=t+1}^{H-1} \gamma^{t+k} w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t}) w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:k})\right]$$

$$= \frac{R_{\max}^2}{N}\left(\sum_{t=0}^{H-1} \gamma^{2t} \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t})^2\right] + 2\sum_{t=0}^{H-2} \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t})^2\right] \sum_{k=t+1}^{H-1} \gamma^{t+k}\right) \tag{6.47}$$

$$= \frac{R_{\max}^2}{N} \sum_{t=0}^{H-1}\left(\gamma^{2t} + \frac{2\gamma^t(\gamma^{t+1} - \gamma^H)}{1-\gamma}\right) \mathop{\mathbb{E}}_{\tau\sim p_{\boldsymbol{\theta}}}\left[w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t})^2\right] \tag{6.48}$$

$$= \frac{R_{\max}^2}{N} \sum_{t=0}^{H-1} \frac{\gamma^t\left(\gamma^t + \gamma^{t+1} - 2\gamma^H\right)}{1-\gamma} d_2\left(p_{\boldsymbol{\theta}'} \| p_{\boldsymbol{\theta}}\right), \tag{6.49}$$

where line (6.44) follows from the fact that the trajectories τ_i are i.i.d., line (6.45) is obtained by bounding the variance with the second moment, line (6.46) is from the fact that the immediate reward is uniformly bounded, line (6.47) is obtained by observing that $\mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}}\left[w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t})w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:k})\right] = \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}}\left[w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t})^2\right]$ as $k > t$, line (6.48) follows from the properties of the geometric sum and finally line (6.49) is obtained from the definition of d_2 and $p_{\boldsymbol{\theta}}$. By taking the limit we get the expression for $\gamma = 1$:

$$\lim_{\gamma\to 1}\frac{\gamma^t\left(\gamma^t+\gamma^{t+1}-2\gamma^H\right)}{1-\gamma} = 2H-2t-1. \tag{6.50}$$

■

Using the estimator defined in Equation (6.41) and the bound on the variance given in Lemma 6.5 we can define the surrogate objective for *per-Decision action-based POIS* (D-POIS):

$$\begin{aligned}\mathcal{L}_{\lambda}^{\mathrm{D-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta}) = &\underbrace{\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H-1}\gamma^t w_{\boldsymbol{\theta}'/\boldsymbol{\theta}}(\tau_{0:t}^{(i)})R(s_t^{(i)},a_t^{(i)})}_{\widehat{J}^{\mathrm{D-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta})}\\ &-\lambda\sqrt{\frac{1}{N}\sum_{t=0}^{H-1}c_t d_2\left(p_{\boldsymbol{\theta}}(\tau_{0:t})\|p_{\boldsymbol{\theta}}(\tau_{0:t})\right)},\end{aligned} \tag{6.51}$$

where $\lambda = R_{\max}\sqrt{(1-\delta)/\delta}$ is the theoretical regularization parameter and c_t is defined in Lemma 6.5. As in the action-based setting, we need to estimate the exponentiated Rényi divergence in practice. We adapt the estimator from Equation (6.38) by limiting the product to time t instead of $H-1$.

Similarly to A-POIS, we can derive a multiple importance sampling extension based on the balance heuristic for each time step $t = 0,\dots,H-1$. Let $\boldsymbol{\theta}_{1:J} = \{\boldsymbol{\theta}_j\}_{j=1}^{J}$ be the set of behavioral policy parameters, and define the per-decision multiple importance weights as:

$$w_{\boldsymbol{\theta}'/\boldsymbol{\theta}_{1:J}}^{\mathrm{BH}}(\tau_{0:t}) = \frac{p_{\boldsymbol{\theta}'}(\tau_{0:t})}{\sum_{k=1}^{J}\frac{N_k}{N}p_{\boldsymbol{\theta}_j}(\tau_{0:t})} = \frac{\prod_{h=0}^{t}\pi_{\boldsymbol{\theta}'}(a_h|s_h)}{\sum_{k=1}^{J}\prod_{h=0}^{t}\frac{N_k}{N}\pi_{\boldsymbol{\theta}_j}(a_h|s_h)}.$$

From Corollary 6.1 and Lemma 6.5, we obtain the following surrogate objective:

$$\begin{aligned}\mathcal{L}_{\lambda}^{\mathrm{D-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta}_{1:J}) = &\underbrace{\frac{1}{N}\sum_{j=1}^{J}\sum_{i=1}^{N}\sum_{t=0}^{H-1}\gamma^t w_{\boldsymbol{\theta}'/\boldsymbol{\theta}_{1:J}}^{\mathrm{BH}}(\tau_{0:t}^{(ij)})R(s_t^{(ij)},a_t^{(ij)})}_{\widehat{J}^{\mathrm{D-POIS}}(\boldsymbol{\theta}'/\boldsymbol{\theta}_{1:J})}\\ &-\lambda\sqrt{\frac{1}{N}\sum_{t=0}^{H-1}\frac{c_t}{\sum_{j=1}^{J}\frac{N_j}{d_2\left(p_{\boldsymbol{\theta}'}(\tau_{0:t})\|p_{\boldsymbol{\theta}_j}(\tau_{0:t})\right)}}},\end{aligned} \tag{6.52}$$

where each τ_{ij} is obtained by running policy $\pi_{\boldsymbol{\theta}_j}$ in the environment with $i = 1, 2, ..., N_j$ and $j = 1, 2, ..., J$. Pseudocode for D-POIS is obtained from Algorithm 7 by replacing all occurrences of $\mathcal{L}_\lambda^{\mathrm{A-POIS}}$ with $\mathcal{L}_\lambda^{\mathrm{D-POIS}}$.

A remark on exponential concentration inequalities. We have based our surrogate objectives on Chebyshev-like inequalities. This results in a polynomial dependence on the (inverse) failure probability $1/\delta$. By using Hoeffding's or related inequalities, we could obtain a logarithmic dependence instead. This is called an *exponential* bound. An exponential lower bound on off-policy performance would result in a less conservative surrogate objective, allowing faster learning with the same theoretical guarantees. Unfortunately, the heavy-tailed nature of importance weights prevents us to achieve exponential concentration even if the reward function is bounded. This limitation can be overcome by employing so-called *robust estimators* (Bubeck et al., 2013). A robust importance-sampling estimator can be obtained by clipping the importance weights. This technique was used by Papini et al. (2019a) to construct a risk-seeking counterpart of the surrogate objective of POIS, that is, a statistical *upper* bound on off-policy performance, for exploration purposes. In that work, exponential concentration was necessary to prove regret guarantees.[5] However, clipping the importance weights makes the surrogate objective non-differentiable. For this reason, this technique is unsuitable to policy-gradient approaches like POIS. Developing differentiable robust estimators is a promising direction for future research on off-policy policy optimization.

6.4 Experiments

In this section, we present the experimental evaluation of POIS in its different flavors (parameter-based, action-based, action-based per-decision). We first provide a set of empirical comparisons on classical continuous control tasks with linearly parametrized policies (Section 6.4.1); we then show how POIS can be adopted for learning deep neural policies (Section 6.4.2). We also study the effects of employing per-decision importance weights and multiple importance weights (Section 6.4.3). In all experiments, for A-POIS and D-POIS we used the IS estimator, while for P-POIS we employed the SN estimator (Equation 6.6). All the tasks are from the `rllab` library (Duan et al., 2016). See Section C.3 for more details.

6.4.1 Linear Policies

Linearly parametrized Gaussian policies can be applied to relatively complex control tasks (Rajeswaran et al., 2017). In this section, we compare the learning performance of A-POIS, D-POIS, and P-POIS against TRPO (Schulman et al., 2015a) and PPO (Schulman et al., 2017b) on some classical continuous control benchmarks (Duan et al., 2016) when all the algorithms use linear policies. The hyperparameters of the individual algorithms are reported in Table 6.1. In the

[5] An adaptive clipping threshold allowed to handle the bias introduced by weight clipping (Papini et al., 2019a).

Task	P-POIS (δ)	A-POIS (δ)	D-POIS (δ)	TRPO (step size)	PPO (step size)
Cartpole	0.4	0.4	0.99	0.1	0.01
Inverted Double Pendulum	0.1	0.1	0.4	0.1	1
Acrobot	0.2	0.7	0.7	1	1
Mountain Car	1	0.9	0.9	0.01	1
Inverted Pendulum	0.8	0.9	0.9999	0.01	0.01

Table 6.1: Meta-parameter value of the individual algorithms employed in the experiments shown in Figure 6.1. For all versions of POIS we report the value of δ, while for TRPO (Schulman et al., 2015a) and PPO (Schulman et al., 2017b) the value of the step size.

Cartpole environment, as we can see from Figure 6.1, all the POIS variants outperform significantly the performance of TRPO and PPO, showing not only the convergence to the optimum but also a faster convergence speed. This is particularly true of P-POIS and D-POIS, where the optimum is reached in very few iterations. Indeed, in this case, we can appreciate the benefit of the PDIS technique over the simple IS. For the Inverted Double Pendulum environment, we have a less consistent behavior: A-POIS has similar performance as TRPO and PPO, while P-POIS finds the optimal policy at a remarkable speed; D-POIS stays somewhere in the middle, still reaching the optimum but at a slower rate. In the acrobot task, we see how, once again, P-POIS outperforms both TRPO and PPO, while A-POIS and D-POIS get stuck in what could be a local optimum. The mountain-car environment shows a very similar behavior among all the benchmarked algorithms, with A-POIS and D-POIS having a slightly slower convergence speed. Lastly, the inverted-pendulum setting is the only environment in which no version of POIS can keep up with the TRPO and PPO baselines. Overall, POIS displays a performance comparable with TRPO and PPO across the tasks. In particular, P-POIS displays better performance w.r.t. A-POIS. Furthermore, apart from the peculiar case of the inverted pendulum, D-POIS performs at least as good as A-POIS, empirically supporting the intuition that the PDIS technique helps to reduce the variance in practice.

In Figure 6.2 we show, for several metrics, the behavior of A-POIS when changing the δ parameter in the Cartpole environment. We can see that when δ is small (e.g., 0.2), the Effective Sample Size (ESS) remains large and, consequently, the variance of the importance weights ($\mathbb{V}\text{ar}[w]$) is small. This means that the penalty term in the objective function discourages the optimization process from selecting policies that are far from the behavioral policy. As a consequence, the displayed behavior is very conservative, preventing the policy from reaching the optimum. On the contrary, when δ approaches 1, the ESS is smaller and the variance of the weights tends to increase significantly. Again, the performance remains suboptimal as the penalty term in the objective function is too small. The best behavior is obtained with an intermediate value of δ, specifically 0.4.

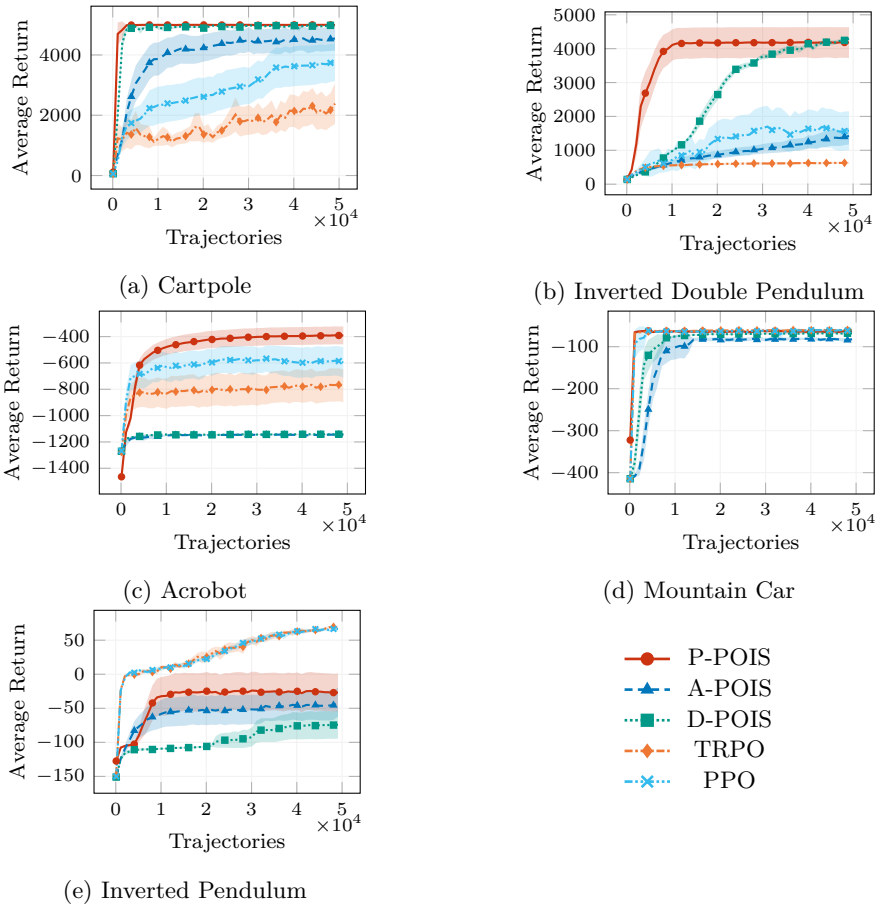


FIGURE 6.1: Average return as a function of the number of trajectories for P-POIS, A-POIS, D-POIS, TRPO, and PPO with *linear policy* (20 runs, 95% confidence intervals).

6.4.2 Deep Neural Policies

In this section, we adopt a deep neural network (3 layers of 100, 50, and 25 neurons) to represent the policy. The experimental setup is fully compatible with the classical benchmark by Duan et al. (2016). The value of the hyperparameters is reported in Table 6.2. While A-POIS and D-POIS can be directly applied to deep neural networks, P-POIS exhibits some critical issues. A high-dimensional hyperpolicy (like a Gaussian from which the weights of a deep neural policy are sampled) can make $d_2(\rho_{\boldsymbol{\nu}'} \| \rho_{\boldsymbol{\nu}})$ extremely sensitive to small parameter changes, which leads to over-conservative updates.[6] A first practical variant comes from the insight that

[6]This *curse of dimensionality* can be seen as the analog, for the parameter-based exploration framework, of the pesky dependence of the Rényi divergence on the task horizon H in the

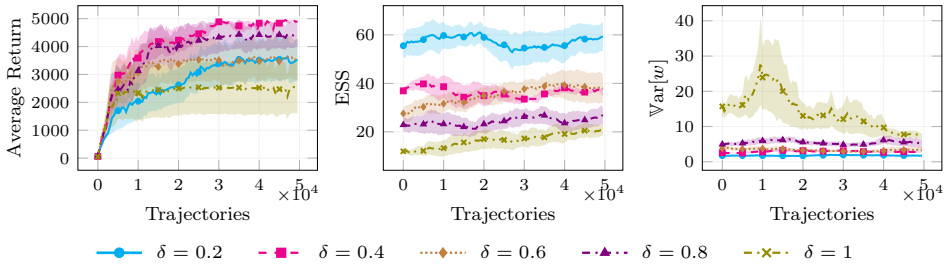


Figure 6.2: Average return, Effective Sample Size (ESS), and variance of the importance weights ($\mathbb{V}\mathrm{ar}[w]$) as a function of the number of trajectories for A-POIS for different values of the parameter δ in the Cartpole environment (20 runs, 95% c.i.).

Task	P-POIS (δ)	A-POIS (δ)	D-POIS (δ)
Inverted Double Pendulum	0.8	0.99	0.4
Cartpole	0.6	0.99	0.99
Mountain Car	0.3	0.99	0.99
Swimmer	0.6	0.99	0.99

Table 6.2: Meta-parameter value of the individual algorithms employed in the experiments shown in Figure 6.3. For all versions of POIS we report the value of δ.

$d_2(\rho_{\boldsymbol{\nu}'} \| \rho_{\boldsymbol{\nu}})/N$ is the inverse of the effective sample size, as reported in Equation 6.7. We can obtain a less conservative (although approximate) surrogate function by replacing it with $1/\widehat{\mathrm{ESS}}(\rho_{\boldsymbol{\nu}'} \| \rho_{\boldsymbol{\nu}})$. Another trick is to model the hyperpolicy as a set of independent Gaussians, each defined over a disjoint subspace of Θ. In Table 6.3, we augmented the results provided in (Duan et al., 2016). All the algorithms are action-based except the last three. CMA-ES (Hansen and Ostermeier, 2001) is an Evolution Strategy (Section 3.12.2) with Covariance Matrix Adaptation. In Figure 6.3, we also provide performance curves for our algorithms, as we did in the previous section. We can see that A-POIS and D-POIS are able to achieve an overall behavior similar to the best of the action-based algorithms, approaching TRPO and beating DDPG. Similarly, P-POIS exhibits performance similar to CEM (Szita and Lörincz, 2006), the best performing among the parameter-based methods.

6.4.3 Multiple P-POIS

We also present some results related to the multiple importance sampling extension we introduced in Section 6.1.3. While this extension can be applied to every flavor of POIS, we only focus on the P-POIS setting with a linear policy, in order to briefly present the pros and cons of this particular strategy. The hyperparameter values are

action-based case.

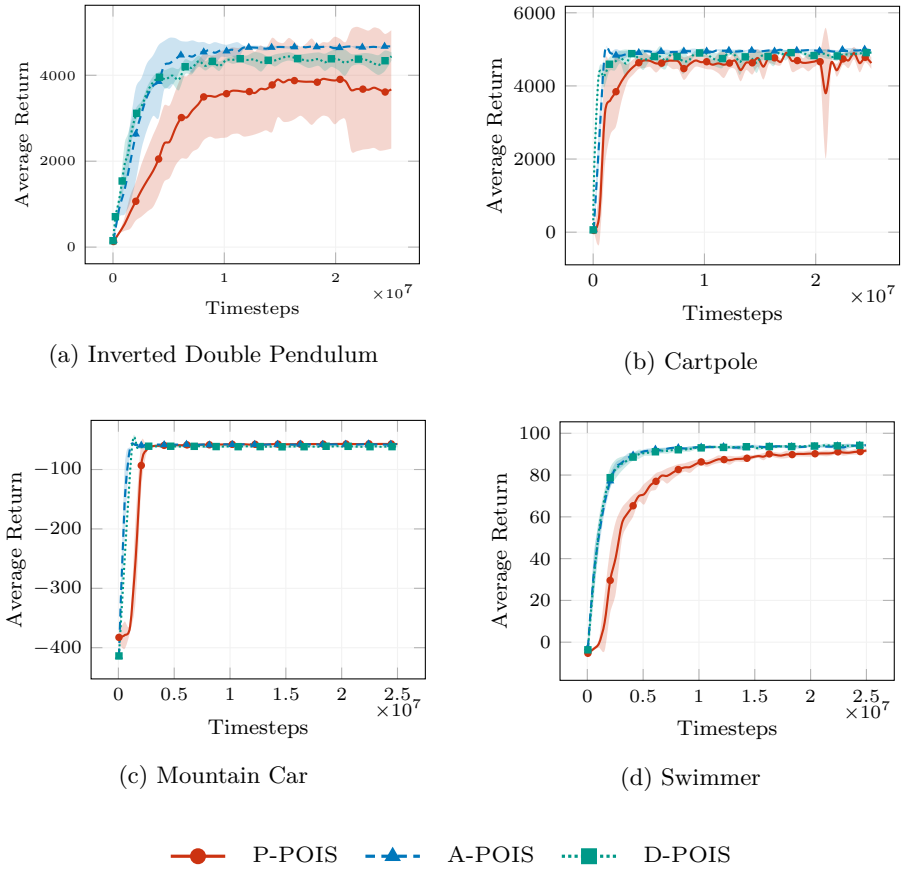


FIGURE 6.3: Average return as a function of the number of trajectories for A-POIS, P-POIS with deep neural policies (5 runs, 95% confidence intervals).

reported in Table 6.4. To highlight the benefits of multiple importance weights on the effective sample size, we set the batch size to one. In the original P-POIS, this means that our agent collects a single trajectory per (on-line) iteration. With MIS, in principle, we could employ all the previous trajectories at each iteration, provided that we store all the past behavioral hyper-policies, together with their sampled policy parameters and the resulting returns. For computational reasons, we employ a finite memory, managed as a simple FIFO queue. The *capacity* of this queue is the number of most recent behavioral hyper-policies it can store (corresponding to J in Equation (6.10)). We use weight normalization whenever possible. For MIS, we employ the self-normalized weights from Equation (6.12). In Figure 6.4 we compare, on the usual benchmark tasks, P-POIS with single importance sampling (i.e., $J = 1$) to its MIS counterpart for different values of the capacity. We also report single-IS P-POIS with a batch size of 10. The latter allows to appreciate the difference

Algorithm	Cart-Pole Balancing	Mountain Car	Double Inverted Pendulum	Swimmer
Random	77.1 ± 0.0	-415.4 ± 0.0	149.7 ± 0.1	-1.7 ± 0.1
REINFORCE	4693.7 ± 14.0	-67.1 ± 1.0	4116.5 ± 65.2	92.3 ± 0.1
TNPG	$\mathbf{3986.4 \pm 748.9}$	$\mathbf{-66.5 \pm 4.5}$	$\mathbf{4455.4 \pm 37.6}$	$\mathbf{96.0 \pm 0.2}$
RWR	$\mathbf{4861.5 \pm 12.3}$	-79.4 ± 1.1	3614.8 ± 368.1	60.7 ± 5.5
REPS	565.6 ± 137.6	-275.6 ± 166.3	446.7 ± 114.8	3.8 ± 3.3
TRPO	$\mathbf{4869.8 \pm 37.6}$	$\mathbf{-61.7 \pm 0.9}$	$\mathbf{4412.4 \pm 50.4}$	$\mathbf{96.0 \pm 0.2}$
DDPG	4634.4 ± 87.6	-288.4 ± 170.3	2863.4 ± 154.0	85.8 ± 1.8
A-POIS	$\mathbf{4842.8 \pm 13.0}$	-63.7 ± 0.5	$\mathbf{4232.1 \pm 189.5}$	88.7 ± 0.55
D-POIS	$\mathbf{4819.3 \pm 59.3}$	$\mathbf{-61.0 \pm 0.5}$	$\mathbf{4333.8 \pm 115.4}$	88.2 ± 1.49
CEM	4815.4 ± 4.8	-66.0 ± 2.4	2566.2 ± 178.9	68.8 ± 2.4
CMA-ES	2440.4 ± 568.3	-85.0 ± 7.7	1576.1 ± 51.3	64.9 ± 1.4
P-POIS	4428.1 ± 138.6	-78.9 ± 2.5	3161.4 ± 959.2	76.8 ± 1.6

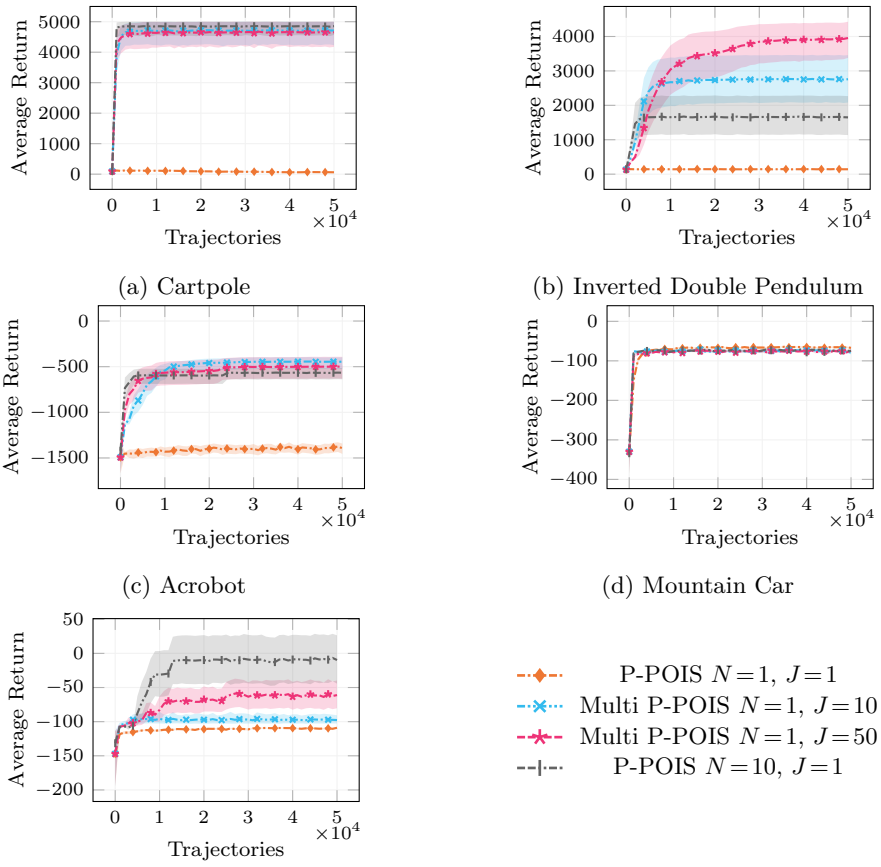
Table 6.3: Performance of POIS compared with Duan et al. (2016) on *deep neural policies* (5 runs, 95% confidence intervals). In **bold**, the performances that are not statistically significantly different from the best algorithm in each task.

Environment	$N = 1$, $J = 1$	$N = 1$, $J = 10$	$N = 1$, $J = 50$	$N = 10$, $J = 1$
Cartpole	0.0001	0.0001	0.001	0.1
Inverted D. P.	0.001	0.001	0.0005	0.05
Acrobot	0.99	0.01	0.05	0.6
Mountain Car	0.6	0.0005	0.0005	0.05
Inverted Pendulum	0.99	0.2	0.1	0.1

Table 6.4: Meta-parameter value of the individual algorithms employed in the experiments shown in Figure 6.4. For all versions of POIS we report the value of δ.

between having 10 "fresh" samples from the current behavioral hyper-policy and re-using old ones with MIS instead.

In all the considered tasks, single P-POIS with unit batch size fails miserably, except in Mountain Car, in which all the tested variants show comparable behavior. We can see that MIS is able to remedy this lack of samples, at least partially. In Cartpole, a capacity of 10 is enough to achieve optimal performance. The results in the Inverted Double Pendulum task are the most aligned with intuition: a capacity of 10 yields a significant improvement compared to the single-IS case, but the latter becomes superior once equipped with a batch size of 10 fresh samples. In addition, a larger capacity ($J = 50$) is beneficial. Acrobot yields similar results, although less clear. The outcomes in the Inverted Pendulum task are more surprising: a capacity of 10 is better than both a capacity of 50 and the large-batch variant. This could be explained by the paramount importance of exploration in this task if we consider the variance of the objective function estimate as a passive form of exploration. Another possibility is that adding more behavioral hyper-policies makes it harder to optimize the objective function. Both aspects should be further inquired by future work.



(a) Cartpole

(b) Inverted Double Pendulum

(c) Acrobot

(d) Mountain Car

(e) Inverted Pendulum

FIGURE 6.4: Average return as a function of the number of trajectories for P-POIS with *linear policy* for different values of the batch size N and the MIS capacity J (20 runs, 95% confidence intervals).

We will discuss other open questions on POIS in the conclusion (Chapter 9).

Convergence Guarantees 7

We have stressed several times the fact that interaction data are a precious resource in real-world reinforcement learning. In this chapter, we study the amount of data that *Policy Gradient* (PG) algorithms require to converge. Besides the inherent theoretical interest, *finite-sample convergence guarantees* provide an important certificate of the time and effort a learning process will require.

Although the asymptotic convergence of policy gradient methods are widely known at least since Sutton et al. (1999), the sample complexity of these algorithms only recently gained attention. The convergence rate of REINFORCE and Monte Carlo PGT can be easily derived from the classic theory of stochastic gradient descent, and may therefore be considered trivial. However, choosing the most meaningful assumptions for the policy class is far from obvious. Moreover, we present here a series of episodic actor-only policy gradient algorithms that have provably better sample complexity than REINFORCE.

The structure of this chapter is as follows. In Section 7.1 we formally introduce our optimization framework with its basic assumptions. In Section 7.2 we provide a proof of the $\mathcal{O}(\epsilon^{-2})$ sample complexity of REINFORCE, which is a folklore result. We also discuss what assumptions on the policy class are really necessary to prove the convergence of REINFORCE and PGT. We then proceed to develop a variant of REINFORCE with a better sample complexity, based on the *Stochastic Variance-Reduced Gradient* (SVRG) technique from finite sum optimization (Johnson and Zhang, 2013). Our algorithm, called *Stochastic Variance-Reduced Policy Gradient* (SVRPG), is presented in Section 7.3. We report a proof by Xu et al. (2019) showing that SVRPG enjoys a $\mathcal{O}(\epsilon^{-5/3})$ sample complexity, which is indeed better than REINFORCE. We also raise some concerns about the strong assumptions on which these kind of convergence proofs rely on. In Section 7.4, we briefly review algorithms that were proposed after SVRPG and that enjoy an even better $\mathcal{O}(\epsilon^{-3/2})$ rate, which is believed to be optimal. We review other related works in Section 7.5. In Section 7.6, we provide an empirical evaluation of SVRPG. To complete our account

of convergence guarantees for policy gradient algorithms, we briefly review recent works on global optimality guarantees in Section 7.7. We conclude by raising some convergence-related concerns on the practice of learning the variance of Gaussian policies with policy gradients, which offer further motivation for Chapter 8.

This chapter is based on our original SVRPG paper (Papini et al., 2018) and includes some later contributions by Xu et al. (2019).

7.1 Optimization Framework

We are interested in the convergence rate of actor-only policy optimization algorithms (Section 3.5). Since these are typically episodic, we will measure their sample complexity in terms of the number of *trajectories* they need to collect in order to converge to a stationary point.

As usual, we want to maximize expected performance (3.1) over a parametric policy class Π_Θ. For simplicity, consider the unconstrained case $\Theta = \mathbb{R}^d$ (see Xu et al., 2020, for a discussion of the constrained case). We say $\boldsymbol{\theta} \in \Theta$ *is at convergence* when it satisfies the *first-order condition* $\nabla J(\boldsymbol{\theta}) = 0$. For any given $\epsilon > 0$, we characterize ϵ-convergence as:

$$\left\|\nabla J(\boldsymbol{\theta})\right\|^2 \leqslant \epsilon. \tag{7.1}$$

We say that a policy gradient algorithm has sample complexity $\mathcal{C} : \mathbb{R}_{++} \to \mathbb{R}$ if the minimum number of trajectories guaranteeing ϵ-convergence is $\mathcal{O}(\mathcal{C}(\epsilon))$. An algorithm with this property also converges asymptotically since (7.1) holds for all $\epsilon > 0$ given enough data.

We assume access to a FSO $\widehat{\nabla} J(\boldsymbol{\theta}; \mathcal{D})$, where $\mathcal{D}$ denotes a dataset of i.i.d. trajectories collected with $\pi_{\boldsymbol{\theta}}$. This can be, for instance, a REINFORCE or G(PO)MDP policy gradient estimator (Section 3.5.1 and 3.5.2, respectively). In the following, let $g(\boldsymbol{\theta}; \tau) = \widehat{\nabla} J(\boldsymbol{\theta}; \{\tau\})$ denote the gradient estimate obtained from a single trajectory. We further require the following:

Assumption 7.1 (PG convergence requirements)

1. *The reward is bounded as* $\|r\|_\infty \leqslant R_{\max}$ *(Assumption 5.1), and* $\gamma < 1$*;*
2. *The policy class* Π_Θ *is* (ξ_1, ξ_2, ξ_3)*-smoothing (see Definition 5.2.1), with* $\Theta = \mathbb{R}^d$*;*
3. *The variance of the gradient estimator is bounded, for all* $\boldsymbol{\theta} \in \Theta$*, as:*

$$\underset{\tau \sim p_{\boldsymbol{\theta}}}{\mathbb{V}ar}\left[g(\boldsymbol{\theta}; \tau)\right] \leqslant \mathcal{V}_1, \tag{7.2}$$

for some $R_{\max}, \xi_1, \xi_2, \xi_3, \mathcal{V}_1 \geqslant 0$.

The first two conditions guarantee that the objective function is smooth. We have shown in Section 5.2 that commonly used policy classes are smoothing. The requirement on the variance of the gradient estimator is common in first-order stochastic optimization (e.g., Allen-Zhu, 2018). As shown in Chapter 5, it is satisfied both by REINFORCE (Lemma 5.4) and by G(PO)MDP (Lemma 5.5).

7.2 Convergence Rate of REINFORCE

It is well known that REINFORCE enjoys the typical $\mathcal{O}(\epsilon^{-2})$ convergence rate of smooth nonconvex SGD under some regularity conditions.[1] However, formal proofs of this result are surprisingly hard to find both in the PO and in the nonconvex optimization literature.[2] In this section, we prove the convergence rate of actor-only policy gradient (Algorithm 2) under Assumption 7.1. As discussed in Section 7.1, this covers interesting practical scenarios.

In the following, let $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\Theta} J(\boldsymbol{\theta})$ denote a vector of globally optimal policy parameters. Note that the iterates $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots$ of Algorithm 2 are random variables, since they are updated using stochastic gradients. Recall that $\mathcal{D}_k$ is the dataset sampled at the k-th iteration. Let $\mathcal{F}_k = \sigma(\bigcup_{i=0}^{k-1} \mathcal{D}_i)$ denote the σ-algebra representing all information available at the beginning of the k-th iteration, and let $\mathbb{E}_k[\cdot]$ be short for $\mathbb{E}_{\mathcal{D}_k\sim p_{\boldsymbol{\theta}_k}}[\cdot|\mathcal{F}_k]$. Now consider the filtration $\mathcal{F} = \{\mathcal{F}_k\}_{k\geqslant 0}$ and let $\{X_k\}_{k\geqslant 0}$ be an $\mathcal{F}$-adapted process (one example is the sequence $\{\boldsymbol{\theta}_k\}_{k\geqslant 0}$ of policy parameters). Let $\mathbb{E}_{0:K}[\cdot]$ be short for $\mathbb{E}_0[\mathbb{E}_1[\dots\mathbb{E}_{K-1}[\mathbb{E}_K[\cdot]]\dots]]$. By the law of total expectation:

$$\sum_{k=0}^{K-1} \mathbb{E}_k[X_{k+1}] = \mathbb{E}_{0:K-1}\left[\sum_{k=0}^{K-1} X_{k+1}\right]. \tag{7.3}$$

Finally, let v_k be short for $\widehat{\nabla} J(\boldsymbol{\theta}_k; \mathcal{D}_k)$.

First, we establish a lower bound for the number of iterations required to achieve ϵ-convergence:

Theorem 7.1 (REINFORCE rate) *Under Assumption 7.1, running Algorithm 2 with initial policy parameters $\boldsymbol{\theta}_0$, batch size N, and step size $\alpha = \min\left\{\frac{1}{L}, \frac{\epsilon N}{L\mathcal{V}_1}\right\}$ for a number K of iterations such that:*

$$K \geqslant 2\left(J(\boldsymbol{\theta}^*) - J(\boldsymbol{\theta}_0)\right)\left(\frac{L}{\epsilon} + \frac{L\mathcal{V}_1}{\epsilon^2 N}\right), \tag{7.4}$$

guarantees $\mathbb{E}\left[\left\|\nabla J(\boldsymbol{\theta}_k)\right\|^2\right] \leqslant \epsilon$ for a k uniformly sampled from $\{0, 1, \dots, K-1\}$, where $L = \frac{R_{\max}}{(1-\gamma)^2}\left(\frac{2\gamma\xi_1^2}{1-\gamma} + \xi_2 + \xi_3\right)$, and all the other constants are from Assumption 7.1.

Proof From Assumption 7.1 and Theorem 5.2, for all $k \geqslant 0$:

$$J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant \langle \nabla J(\boldsymbol{\theta}_k), \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\rangle - \frac{L}{2}\left\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\right\|^2. \tag{7.5}$$

[1] Some authors (e.g., Pham et al., 2020) define ϵ-convergence as $\|\nabla J(\boldsymbol{\theta})\| \leqslant \epsilon$, without the square. Albeit conceptually equivalent, it produces different convergence rates. For instance, the one for REINFORCE is $\mathcal{O}(\epsilon^{-4})$ under this alternative definition. To avoid any confusion in comparing algorithms, we always use (7.1) in this manuscript.

[2] This is what is called a *folklore* result. Our proof for REINFORCE follows the one by Allen-Zhu (2018, Appendix B) for SGD.

Hence:

$$\begin{aligned}
\mathbb{E}_k\left[J(\boldsymbol{\theta}_{k+1})\right] - J(\boldsymbol{\theta}_k) &\geqslant \mathbb{E}_k\left[\langle \nabla J(\boldsymbol{\theta}_k), \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\rangle - \frac{L}{2}\left\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\right\|^2\right] \\
&= \alpha\left\|\nabla J(\boldsymbol{\theta}_k)\right\|^2 - \frac{\alpha^2 L}{2}\mathbb{E}_k\left[\left\|v_k\right\|^2\right] \\
&\geqslant \left(\alpha - \frac{\alpha^2 L}{2}\right)\left\|\nabla J(\boldsymbol{\theta}_k)\right\|^2 - \frac{\alpha^2 L}{2}\mathbb{E}_k\left[\left\|v_k - \nabla J(\boldsymbol{\theta}_k)\right\|^2\right] \qquad (7.6) \\
&\geqslant \left(\alpha - \frac{\alpha^2 L}{2}\right)\left\|\nabla J(\boldsymbol{\theta}_k)\right\|^2 - \frac{\alpha^2 L}{2N}\mathcal{V}_1, \qquad (7.7)
\end{aligned}$$

where (7.6) is from the triangle inequality and (7.7) is from Assumption 7.1.3. Consider the following sum:

$$\begin{aligned}
\sum_{k=0}^{K-1}\mathbb{E}_k\left[J(\boldsymbol{\theta}_{k+1})\right] - J(\boldsymbol{\theta}_k) &= \sum_{k=0}^{K-1}\mathbb{E}_k\left[J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k)\right] \\
&= \mathbb{E}_{0:K-1}\left[\sum_{k=0}^{K-1} J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k)\right] \qquad (7.8) \\
&= \mathbb{E}_{0:K-1}\left[J(\boldsymbol{\theta}_K) - J(\boldsymbol{\theta}_0)\right] \qquad (7.9) \\
&= \mathbb{E}_{0:K-1}\left[J(\boldsymbol{\theta}_K)\right] - J(\boldsymbol{\theta}_0),
\end{aligned}$$

where (7.8) is from (7.3) and (7.9) is by telescoping the sum. So, summing both sides of (7.7) over $k = 0, 1, \ldots, K-1$ and dividing by K:

$$\frac{\mathbb{E}_{0:K-1}\left[J(\boldsymbol{\theta}_K)\right] - J(\boldsymbol{\theta}_0)}{K} \geqslant \frac{\left(\alpha - \frac{\alpha^2 L}{2}\right)}{K}\sum_{k=0}^{K-1}\left\|\nabla J(\boldsymbol{\theta}_k)\right\|^2 - \frac{\alpha^2 L}{2N}\mathcal{V}_1. \qquad (7.10)$$

By rearranging terms:

$$\frac{1}{K}\sum_{k=0}^{K-1}\left\|\nabla J(\boldsymbol{\theta}_k)\right\|^2 \leqslant \frac{\frac{\mathbb{E}_{0:K-1}[J(\boldsymbol{\theta}_K)] - J(\boldsymbol{\theta}_0)}{K} + \frac{\alpha^2 L}{2N}\mathcal{V}_1}{\left(\alpha - \frac{\alpha^2 L}{2}\right)}. \qquad (7.11)$$

The term on the left hand side is the expected value of the squared gradient norm w.r.t. to an iterate randomly chosen from $k = 0$ to $k = K-1$. For convergence, we require this quantity to be less than some error ϵ. By combining (7.11) with (7.1) and rearranging terms, we obtain the following lower bound on the number K of iterations sufficient to achieve convergence:

$$K \geqslant \frac{\mathbb{E}_{0:K-1}\left[J(\boldsymbol{\theta}_K)\right] - J(\boldsymbol{\theta}_0)}{\epsilon\alpha - \epsilon\alpha^2\frac{L}{2} - \alpha^2\frac{L}{2N}\mathcal{V}_1}. \qquad (7.12)$$

We actually require the following:

$$K \geqslant \frac{J(\boldsymbol{\theta}^*) - J(\boldsymbol{\theta}_0)}{\epsilon\alpha - \epsilon\alpha^2\frac{L}{2} - \alpha^2\frac{L}{2N}\mathcal{V}_1}, \qquad (7.13)$$

which is stronger since $\boldsymbol{\theta}^*$ is a global optimizer. We now set as a step size $\alpha = \min\left\{\frac{1}{L}, \frac{\epsilon N}{L\mathcal{V}_1}\right\}$. This allows to consider two cases:

1. $\epsilon \geqslant \mathcal{V}_1/N$
   In this case, $\alpha = \frac{1}{L}$, hence the requirement becomes:
$$K \geqslant \frac{2L\left(J(\boldsymbol{\theta}^*) - J(\boldsymbol{\theta}_0)\right)}{\epsilon - \frac{\mathcal{V}_1}{N}} \geqslant \frac{2L\left(J(\boldsymbol{\theta}^*) - J(\boldsymbol{\theta}_0)\right)}{\epsilon}. \tag{7.14}$$
2. $\epsilon < \mathcal{V}_1/N$
   In this case, $\alpha = \frac{\epsilon N}{L\mathcal{V}_1}$, hence the requirement becomes:
$$K \geqslant \frac{2L\left(J(\boldsymbol{\theta}^*) - J(\boldsymbol{\theta}_0)\right)}{\frac{\epsilon^2 N}{2L\mathcal{V}_1}\left(1 - \frac{\epsilon N}{\mathcal{V}_1}\right)} \geqslant \frac{2L\mathcal{V}_1\left(J(\boldsymbol{\theta}^*) - J(\boldsymbol{\theta}_0)\right)}{\epsilon^2 N}. \tag{7.15}$$

Overall, with $\alpha = \min\left\{\frac{1}{L}, \frac{\epsilon N}{L\mathcal{V}_1}\right\}$, a sufficient number of iterations is:

$$K \geqslant 2L\left(J(\boldsymbol{\theta}^*) - J(\boldsymbol{\theta}_0)\right)\left(\frac{1}{\epsilon} + \frac{\mathcal{V}_1}{\epsilon^2 N}\right), \tag{7.16}$$

which completes the proof. ■

Since N trajectories are collected at each iteration, the minimum number of trajectories required to achieve ϵ-convergence is:

$$N_{\mathrm{TOT}} = KN = \mathcal{O}\left(\left(J(\boldsymbol{\theta}^*) - J(\boldsymbol{\theta}_0)\right)\left(\frac{LN}{\epsilon} + \frac{L\mathcal{V}_1}{\epsilon^2}\right)\right). \tag{7.17}$$

The second term dominates, yielding the well-known $\mathcal{O}(\epsilon^{-2})$ rate.[3]

Theorem 7.1 can be instantiated for Gaussian and Softmax policies by using the value of the smoothness constant L from Table 5.1. The policy gradient estimator can be REINFORCE or G(PO)MDP, and the corresponding value of $\mathcal{V}_1$ can be deduced from Lemma 5.4 and 5.5, respectively. Note that the choice of batch size does not affect the rate of convergence. By sending N to infinity, we can make the $\mathcal{O}(\epsilon^{-2})$ term from (7.4) vanish, matching the $\mathcal{O}(\epsilon^{-1})$ rate of *exact* gradient descent for smooth nonconvex objectives (Nesterov, 2004). However, this is vacuous in terms of sample complexity. Improving the latter requires novel algorithmic ideas such as the ones described in the next sections.

7.2.1 Remark on the convergence of PGT

As observed in Section 3.1, the original convergence guarantees for PGT (Sutton et al., 1999, Theorem 3), rely on a uniform upper bound over $\left\|\nabla^2 \pi_{\boldsymbol{\theta}}(a|s)\right\|$, which is

[3] As observed by Allen-Zhu (2018), acceleration techniques can only improve the $O(\epsilon^{-1})$ term.

problematic even for the simple shallow Gaussian policy (cf. Equation 3.63), since actions sampled from it are potentially unbounded. This can be easily overcome by replacing the original condition with a smoothing assumption. By inspecting the proof of Theorem 3 by Sutton et al. (1999), we see that the requirement on the policy Hessian is only needed to ensure that the performance is smooth. So, from Theorem 5.1, it is enough to assume that the policy class is smoothing (Definition 5.2.1). The proof then proceeds as before, using Proposition 3.5 by Bertsekas and Tsitsiklis (1996) to prove convergence to a stationary point. Recall, from Section 3.3.4, that Gaussian policies are smoothing, so we no longer need to restrict the action space $\mathcal{A}$ to apply the convergence results from Sutton et al. (1999).

7.3 Stochastic Variance-Reduced Policy Gradient

In this section, we present our SVRPG algorithm (Papini et al., 2018), which has an improved $\mathcal{O}(\epsilon^{-5/3})$ convergence rate (Xu et al., 2019). First, we revise the SVRG algorithm from finite-sum optimization (Johnson and Zhang, 2013).

7.3.1 Stochastic Variance-Reduced Gradient

Finite-sum optimization is the problem of minimizing an objective function $f : \mathcal{X} \to \mathbb{R}$ which can be decomposed into the sum or average of a finite number of functions $f_i : \mathcal{X} \to \mathbb{R}$:

$$\min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} f_i(\boldsymbol{x}). \tag{7.18}$$

This kind of problem is very common in machine learning. Empirical risk minimization can be cast as a finite-sum optimization problem where each f_i corresponds to a data sample z_i from a dataset $\mathcal{D}$ of size N. In this case, we have $f_i(\boldsymbol{x}) = l(z_i|\boldsymbol{x})$ for some loss l. The original requirement by Johnson and Zhang (2013) is that all f_i must be convex in $\boldsymbol{x}$, and f strongly convex. We focus here on the non-convex case, and only assume all the f_i are smooth functions. Under this hypothesis, *full Gradient Descent* (GD) has a $\mathcal{O}(N\epsilon^{-1})$ sample complexity (Nesterov, 2004). The algorithm, first introduced by Cauchy (1847), is simply an iteration of the following update rule:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \nabla f(\boldsymbol{x}) = \boldsymbol{x} - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(\boldsymbol{x}), \tag{7.19}$$

where $\alpha > 0$ is a learning rate. Each update requires N gradient computations, which can be prohibitively expensive for large values of N (e.g., datasets with millions of entries). *Stochastic Gradient Descent* (SGD) (Robbins and Monro, 1951; Bottou and LeCun, 2003) overcomes this problem by picking a single f_i per iteration at random:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \alpha \nabla f_i(\boldsymbol{x}) \qquad \text{where } i \sim \mathcal{U}([N]). \tag{7.20}$$

This removes the linear dependence on N, but the variance introduced by the randomization yields a worse, $\mathcal{O}(\epsilon^{-2})$ rate (Allen-Zhu, 2018).

Starting from SAG (Stochastic Average Gradient, Le Roux et al., 2012), a series of variations to SGD have been proposed to achieve a better trade-off between convergence speed and per-iteration costs: *Stochastic Variance-Reduced Gradient* (SVRG) (Johnson and Zhang, 2013), S2GD (Semi-Stochastic Gradient Descent, Konecný and Richtárik, 2013), SAGA (Defazio et al., 2014a), Finito (Defazio et al., 2014b), MISO (Minimization by Incremental Surrogate Optimization, Mairal, 2015), and, more recently, SARAH (Nguyen et al., 2017), SPIDER (Fang et al., 2018), SNVRG (Stochastic Nested Variance-Reduced Gradient, Zhou et al., 2018), and STORM (Cutkosky and Orabona, 2019). The common idea is to reuse past gradient computations to reduce the variance of the latest stochastic estimate. In particular, SVRG (or later improvements) is often preferred to other methods for its limited storage requirements, which is a significant advantage when deep neural networks are employed. The extension of SVRG to non-convex smooth objectives presented here was studied by Reddi et al. (2016) and concurrently by Allen-Zhu and Hazan (2016).

The idea of SVRG is to alternate full and stochastic gradient updates. Each $m = O(N)$ iterations, a snapshot $\widetilde{\boldsymbol{x}}$ of the current parameter is saved together with its full gradient $\nabla f(\widetilde{\boldsymbol{x}}) = \frac{1}{N}\sum_{i=1}^{N} \nabla f_i(\widetilde{\boldsymbol{x}})$. In between snapshots, the parameter is updated as $\boldsymbol{x} \leftarrow \boldsymbol{x} + \alpha \blacktriangledown f(\boldsymbol{x})$, with a corrected semi-stochastic gradient defined as:

$$\blacktriangledown f(\boldsymbol{x}) = \underbrace{\nabla f(\widetilde{\boldsymbol{x}})}_{\text{full gradient}} + \underbrace{\nabla f_i(\boldsymbol{x})}_{\text{stochastic gradient}} - \underbrace{\nabla f_i(\widetilde{\boldsymbol{x}})}_{\text{correction}} \qquad \text{where } i \sim \mathcal{U}(N), \tag{7.21}$$

and $\widetilde{\boldsymbol{x}}$ is the most recent snapshot. The corrected gradient $\blacktriangledown f(\boldsymbol{x})$ is unbiased:

$$\mathop{\mathbb{E}}_{i\sim\mathcal{U}(N)} \left[\blacktriangledown f(\boldsymbol{x})\right] = \nabla f(\widetilde{\boldsymbol{x}}) + \nabla f(\boldsymbol{x}) - \nabla f(\widetilde{\boldsymbol{x}}) = \nabla f(\boldsymbol{x}), \tag{7.22}$$

and uses the full gradient as a stabilizer to keep the variance under control. An informal but illuminating demonstration of this fact is the following: say the iterate $\boldsymbol{x}$ is converging to $\boldsymbol{x}^*$. Then:

$$\lim_{\boldsymbol{x}\to\boldsymbol{x}^*} \mathbb{V}\mathrm{ar}\left[\blacktriangledown f(\boldsymbol{x})\right] = \lim_{\boldsymbol{x}\to\boldsymbol{x}^*} \mathbb{V}\mathrm{ar}\left[\nabla f_i(\boldsymbol{x}) - \nabla f_i(\widetilde{\boldsymbol{x}})\right] = 0, \tag{7.23}$$

since also the snapshot $\widetilde{\boldsymbol{x}}$ must converge to $\boldsymbol{x}^*$.[4]

We report the pseudocode for SVRG in Algorithm 8. Each snapshot corresponds to an *epoch*, and we call *iterations* the parameter updates that happen in between. For the iterates, superscripts count epochs and subscripts count inner iterations within each epoch. Note that the first iteration ($k = 0$) of each epoch s corresponds to a full gradient step ($\blacktriangledown f(\boldsymbol{x}_0^s) = \nabla f(\widetilde{\boldsymbol{x}}^s)$) since $\boldsymbol{x}_0^s := \widetilde{\boldsymbol{x}}^s$. The final randomization over the returned iterate is for technical reasons only. In practice, one just returns the latest value.

[4] Note that the correlation between the stochastic-gradient and the correction terms is fundamental, otherwise their variances would just sum up.

Algorithm 8 SVRG (Stochastic Variance-Reduced Gradient)

Input: epoch size $m = O(N)$, step size α, initial parameters $\widetilde{\boldsymbol{x}}^0$, number of epochs S
for $s = 0, 1, \dots, S-1$ **do**
  Compute full gradient $\widetilde{\mu} = \nabla f(\widetilde{\boldsymbol{x}}^s)$
  $\boldsymbol{x}_0^s = \widetilde{\boldsymbol{x}}^s$
  for $k = 0, 1, \dots, m-1$ **do**
    $i \sim \mathcal{U}([N])$
    Compute $\blacktriangledown f(\boldsymbol{x}_k^s) = \widetilde{\mu} + \nabla f_i(\boldsymbol{x}_k^s) - \nabla f_i(\widetilde{\boldsymbol{x}}^s)$
    $\boldsymbol{x}_{k+1}^s = \boldsymbol{x}_k^s - \alpha \blacktriangledown f(\boldsymbol{x}_k^s)$
  end for
  Take snapshot $\widetilde{\boldsymbol{x}}^{s+1} = \boldsymbol{x}_m^s$
end for
return $\boldsymbol{x}_k^s$ with (k, s) uniformly sampled from $\{0, 1, \dots, m-1\} \times \{0, 1, \dots, S-1\}$

Since the full gradient is computed only once in $m = O(N)$ iterations, the *amortized* per-iteration cost of SVRG is only twice that of SGD. This, combined with the variance-reduction property, allows to achieve a sample complexity of $\mathcal{O}(N^{2/3}\epsilon^{-1})$, with the same dependency on ϵ of GD but a better dependency on N (Reddi et al., 2016; Allen-Zhu and Hazan, 2016). This result was recently improved to $\mathcal{O}(\sqrt{N}\epsilon^{-1})$ by Fang et al. (SPIDER, 2018) and Zhou et al. (SNVRG, 2018).

7.3.2 From finite-sum to policy optimization

Stochastic variance-reduction techniques like SVRG require to design control variates that are correlated with stochastic gradients. This only possible in certain stochastic optimization settings. In finite-sum optimization, the original setting of SVRG, this correlation is guaranteed by the artificial process of subsampling from a finite dataset. Fortunately, policy optimization provides a similar correlation mechanism: the natural sampling of trajectories arising from the interaction of the agent with the environment. Still, if we wish to apply SVRG to policy optimization, we need to overcome three main challenges:

Nonconvexity. While SVRG was initially designed for convex objectives, policy optimization is, in general, non convex (more properly *non-concave*, since we have defined it as a maximization problem). As discussed in Section 7.3.1, SVRG also works in the nonconvex case (Reddi et al., 2016; Allen-Zhu and Hazan, 2016) under a smoothness assumption. Smoothness in policy optimization has been discussed in Chapter 5 and can be guaranteed for common stochastic policies and gradient estimators. As shown in Lemma 5.1, the smoothness requirement is always satisfied under Assumption 7.1, the same we adopted to prove the convergence rate of REINFORCE (Theorem 7.1).[5]

[5]For technical reasons, we will adopt a stronger assumption in Section 7.3.4.

Infinite support. Policy optimization cannot be cast as a finite-sum problem. We can define base functions as trajectory returns, but the number of possible trajectories is infinite. The main consequence is that we can no longer compute full gradients exactly: at most, we can estimate them with a large number of trajectories. This means that the variance of the full gradient must also be taken into consideration, as for REINFORCE (Theorem 7.1). This problem is not specific of RL: in supervised learning, it corresponds to the common situation in which the data distribution has infinite support. The problem of inexact full-gradient computations in SVRG was studied by Babanezhad et al. (2015) and Lei and Jordan (2017). The latter proposed the SCSG (Stochastically Controlled Stochastic Gradient) algorithm, which is a variant of SVRG where the full gradient is estimated from a finite batch of data and the epoch length is sampled from a geometric distribution. As later shown by Lei et al. (2017), SCSG has complexity $\mathcal{O}(\epsilon^{-5/3})$ in the smooth nonconvex case with infinite support. [6] This is an improvement of $\epsilon^{1/3}$ over SGD in the same setting. Recently, it was improved by another $\epsilon^{1/6}$ by Zhou et al. (SPIDER, 2018) and Fang et al. (SNVRG, 2018), reaching $\mathcal{O}(\epsilon^{-3/2})$ complexity. As shown by Arjevani et al. (2019), this rate is optimal.

Covariate shift. Since the policy optimization objective is an expectation under the trajectory distribution induced by the current policy (3.69), stochastic gradients are no longer the result of uniform sampling. Even worse, the sampling distribution itself changes over time as we update policy parameters. This *covariate shift* phenomenon is more specific of policy optimization and, to the best of our knowledge, we were the first to study it in combination with stochastic variance reduction (Papini et al., 2018). The immediate consequence is that the correction term from (7.21) may introduce some bias, since it is evaluated in the snapshot with data from an updated parameter. This is an off-policy learning problem and can be solved via *Importance Sampling* (IS) if one can tame the extra variance introduced by importance weights (see Chapter 6), which may itself hamper convergence speed.

In the next section we describe an adaptation of SVRG to policy optimization that is able to overcome these challenges obtaining a better sample complexity than REINFORCE.

7.3.3 SVRPG: the algorithm

Stochastic Variance-Reduced Policy Gradient (SVRPG) (Papini et al., 2018) is an actor-only episodic policy gradient algorithm that uses the same variance-reduction technique as SVRG in order to improve sample complexity. The pseudocode for SVRPG is reported in Algorithm 9. The setting is precisely the one described in Section 7.1. Compared to Section 7.3.1, the iterates are policy parameters $\boldsymbol{\theta} \in \Theta$, the objective function is performance $J(\boldsymbol{\theta})$ and we are maximizing instead of minimizing. As mentioned in Section 7.3.2, the number of possible data points (i.e.,, trajectories)

[6] When we first proposed SVRPG (Papini et al., 2018) we were not aware of the results on non-convex SCSG.

is infinite and the distribution of stochastic gradients is not under our control, but is a function of the current policy parameters. The main difference w.r.t. Algorithm 8 is in the definition of the semi-stochastic gradient:

$$\blacktriangledown J(\boldsymbol{\theta}) = \underbrace{\widehat{\nabla} J(\widetilde{\boldsymbol{\theta}}; \widetilde{\mathcal{D}})}_{\text{full gradient}} + \underbrace{\widehat{\nabla} J(\boldsymbol{\theta}; \mathcal{D})}_{\text{stochastic gradient}} \underbrace{- \frac{1}{B} \sum_{\tau \in \mathcal{D}} w_{\widetilde{\boldsymbol{\theta}}/\boldsymbol{\theta}}(\tau) g(\widetilde{\boldsymbol{\theta}}; \tau)}_{\text{correction}}, \tag{7.24}$$

where $\widetilde{\boldsymbol{\theta}}$ is the most recent snapshot, $\widetilde{\mathcal{D}}$ is a set of N trajectories collected with snapshot policy $\pi_{\widetilde{\boldsymbol{\theta}}}$, $\mathcal{D}$ is a set of $B \ll N$ trajectories collected with current policy $\pi_{\boldsymbol{\theta}}$, $w_{\widetilde{\boldsymbol{\theta}}/\boldsymbol{\theta}}$ is a trajectory-wise importance weight (Equation 6.35) where the current policy plays the role of the behavioral, and $g(\boldsymbol{\theta}; \tau)$ is short for $\widehat{\nabla} J(\boldsymbol{\theta}; \{\tau\})$. Let us comment on the three terms of (7.24) in more detail:

- ◇ The *full gradient* cannot be computed exactly and is estimated with a batch of N trajectories sampled from $p_{\widetilde{\boldsymbol{\theta}}}$, where $\widetilde{\boldsymbol{\theta}}$ is the latest snapshot. Differently from the empirical risk minimization framework, here N is a meta-parameter and not the actual size of the data domain, which is infinite. This term corresponds to a REINFORCE or G(PO)MDP gradient estimator.

- ◇ The *stochastic gradient* is computed from a *mini-batch* of $B \ll N$ trajectories sampled from $p_{\boldsymbol{\theta}}$. Again, this is just REINFORCE/G(PO)MDP with a smaller batch size. The original SVRG used a single data point to compute the stochastic gradient, which corresponds to $B = 1$. Larger mini-batches were first studied by Babanezhad et al. (2015) and Konecný et al. (2016). We use mini-batches essentially to counteract the variance introduced by importance sampling.

- ◇ The *correction* term is an off-policy gradient estimate like the ones described in Section 3.5.5. Importance weights are needed since the gradient is evaluated in snapshot parameters $\widetilde{\boldsymbol{\theta}}$, but uses data sampled with the current policy $\pi_{\boldsymbol{\theta}}$. The latter is necessary to correlate the correction with the stochastic-gradient term, which is fundamental for variance reduction (cf. Equation 7.23).[7]

7.3.4 Convergence rate

To prove convergence results for SVRPG we need stronger assumptions than the ones employed for REINFORCE:

Assumption 7.2 (SVRPG convergence requirements)

1. *The reward is bounded as* $\|r\|_\infty \leqslant R_{\max}$ *(Assumption 5.1), and* $\gamma < 1$*;*

[7] The temporal direction of these importance weights is the opposite of the usual one. Typically, in off-policy PG (Section 3.5.5) and related methods (Chapter 6), data collected in the past are re-weighted to evaluate a more recent policy. Here, data collected with the latest policy are re-weighted to evaluate an old one (the snapshot). Note that SVRPG is an on-policy algorithm.

Algorithm 9 SVRPG (Stochastic Variance-Reduced Policy Gradient)

Input: epoch length m, step size α, batch size N, mini-batch size B, initial parameters $\widetilde{\boldsymbol{\theta}}^0$, number of epochs S
for $s = 0, 1, \dots, S-1$ **do**
  Collect batch $\widetilde{\mathcal{D}} \sim p_{\widetilde{\boldsymbol{\theta}}^s}$ of N trajectories
  Compute full gradient $\widetilde{\mu} = \widehat{\nabla} J(\widetilde{\boldsymbol{\theta}}; \widetilde{\mathcal{D}})$ ▷ $\widetilde{\mathcal{D}}$ can be discarded afterwards
  $\boldsymbol{\theta}_0^s = \widetilde{\boldsymbol{\theta}}^s$
  for $k = 0, 1, \dots, m-1$ **do**
    Collect mini-batch $\mathcal{D} \sim p_{\boldsymbol{\theta}_k^s}$ of B trajectories
    Compute $\blacktriangledown J(\boldsymbol{\theta}_k^s) = \widetilde{\mu} + \widehat{\nabla} J(\boldsymbol{\theta}_k^s; \mathcal{D}) - \frac{1}{B} \sum_{\tau \in \mathcal{D}} w_{\widetilde{\boldsymbol{\theta}}^s / \boldsymbol{\theta}_k^s}(\tau) g(\widetilde{\boldsymbol{\theta}}^s; \tau)$
    $\boldsymbol{\theta}_{k+1}^s = \boldsymbol{\theta}_k^s + \alpha \blacktriangledown J(\boldsymbol{\theta}_k^s)$
  end for
  Take snapshot $\widetilde{\boldsymbol{\theta}}^{s+1} = \boldsymbol{\theta}_m^s$
end for
return $\boldsymbol{\theta}_k^s$ with (k, s) uniformly sampled from $\{0, 1, \dots, m-1\} \times \{0, 1, \dots, S-1\}$

2. *The policy class* Π_Θ *is such that, for all* $\boldsymbol{\theta} \in \Theta$, $s \in \mathcal{S}$, *and* $a \in \mathcal{A}$:

$$\|\nabla \log \pi_{\boldsymbol{\theta}}(a|s)\| \leqslant G_1, \qquad \left\|\nabla^2 \log \pi_{\boldsymbol{\theta}}(a|s)\right\| \leqslant G_2, \tag{7.25}$$

3. *Gradient estimator* $g(\boldsymbol{\theta}; \tau)$ *is REINFORCE or G(PO)MDP.*

4. *The variance of importance weights is bounded as:*

$$\mathbb{V}ar\left[w_{\boldsymbol{\theta}/\boldsymbol{\theta}'}(\tau)\right] \leqslant \mathcal{V}_2, \tag{7.26}$$

for all $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta$ *and trajectories* $\tau \in \mathrm{supp}(p_{\boldsymbol{\theta}'})$.

for some $R_{\max}, G_1, G_2, \mathcal{V}_2 \geqslant 0$.

Under this assumption, we can prove a number of useful results[8]:

Lemma 7.1 *Under Assumption 7.2:*

1. *Performance* J *is* L*-smooth with* $L = \frac{R_{\max}}{(1-\gamma^2)} \left(\frac{1+\gamma}{1-\gamma} G_1^2 + G_2 \right)$;

2. $\mathbb{V}ar\left[g(\boldsymbol{\theta}; \tau)\right] \leqslant \mathcal{V}_1$, *where* $\mathcal{V}_1$ *can be deduce from Table 7.1 depending on whether REINFORCE or G(PO)MDP is used as an estimator;*

3. *For all* $\boldsymbol{\theta} \in \Theta$ *and* $\tau \in \mathrm{supp}(p_{\boldsymbol{\theta}})$, $\|g(\boldsymbol{\theta}; \tau)\| \leqslant C_g$, *where* $C_g = \frac{HG_1 R_{\max}(1-\gamma^H)}{1-\gamma}$.

4. *For all* $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta$ *and* $\tau \in \mathrm{supp}(p_{\boldsymbol{\theta}}) \wedge \mathrm{supp}(p_{\boldsymbol{\theta}'})$, $\|g(\boldsymbol{\theta}; \tau) - g(\boldsymbol{\theta}'; \tau)\| \leqslant L_g \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|$, *where* $L_g = \frac{HG_2 R_{\max}(1-\gamma^H)}{1-\gamma}$;

[8] These results can be found as a series ancillary lemmas in the supplementary materials by Papini et al. (2018), or more compactly as Proposition 5.2 by Xu et al. (2019). The constants may slightly change.

Table 7.1: Upper bound on the variance of policy gradient estimators, based on a single trajectory, for a (G_1, G_1^2, G_2)-smoothing policy.

	REINFORCE	**GPOMDP**
$\mathcal{V}_1$	$\frac{HG_1^2R_{\max}^2(1-\gamma^H)^2}{(1-\gamma)^2}$	$\frac{G_1^2R_{\max}^2\left(1-\gamma^H\right)}{(1-\gamma)^3}$

Proof From Definition 5.2.1 and Assumption 7.2.2, the policy class is clearly (G_1, G_1^2, G_2)-smoothing. So, the smoothness of J is from Theorem 5.1.

For the same reason, the variance upper bound is given by Lemma 5.4 (for REINFORCE) or Lemma 5.5 (for G(PO)MDP).

We prove the last two properties for the REINFORCE estimator. The proof for G(PO)MDP is similar after rewriting it in the PGT form (Equation 3.81).

First, we bound the norm of the estimator:

$$\begin{aligned}\|g(\boldsymbol{\theta};\tau)\| &= \left\|\sum_{t=0}^{H-1}\nabla\log\pi_{\boldsymbol{\theta}}(a_t|s_t)R(\tau)\right\| \\ &\leqslant \frac{R_{\max}(1-\gamma^H)}{1-\gamma}\sum_{t=0}^{H-1}\|\nabla\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\| \\ &\leqslant \frac{HG_1R_{\max}(1-\gamma^H)}{1-\gamma} \qquad (7.27)\end{aligned}$$

Finally, we prove that $g(\cdot;\tau)$ is L_g-smooth by bounding its gradient in norm:

$$\begin{aligned}\|\nabla g(\boldsymbol{\theta};\tau)\| &\leqslant \left\|\sum_{t=0}^{H-1}\nabla^2\log\pi_{\boldsymbol{\theta}}(a_t|s_t)R(\tau)\right\| \\ &\leqslant \frac{R_{\max}(1-\gamma^H)}{1-\gamma}\sum_{t=0}^{H-1}\left\|\nabla^2\log\pi_{\boldsymbol{\theta}}(a_t|s_t)\right\| \\ &\leqslant \frac{HG_2R_{\max}(1-\gamma^H)}{1-\gamma}. \qquad (7.28)\end{aligned}$$

■

From the proof of Lemma 7.1, it shoud be clear that Assumption 7.2 is indeed stronger than Assumption 7.1. Hence, the convergence result proven for REINFORCE also holds under these stronger requirements. We discuss whether the latter are reasonable in Section 7.3.5.

In our original paper (Papini et al., 2018), we only provided asymptotic convergence guarantees for SVRPG. In particular, we showed that, under Assumption 7.2 and an appropriate choice of meta-parameters, the output $\boldsymbol{\theta}_{\text{OUT}}$ of Algorithm 9 satisfies:

$$\mathbb{E}\left[\left\|\nabla J(\boldsymbol{\theta}_{\text{OUT}})^2\right\|\right] \leqslant \frac{J(\boldsymbol{\theta}^*)-J(\boldsymbol{\theta}_0)}{c_1T}+\frac{c_2\mathcal{V}_1}{N}+\frac{c_3L_{\mathbb{V}}}{B}, \qquad (7.29)$$

for some constants $c_1, c_2, c_3 > 0$. This is similar to REINFORCE (cf. Inequality 7.10), with an additional $\mathcal{O}(B^{-1})$ term that is due to the variance introduced by importance weights. This term prevents from establishing a better convergence rate than the trivial $\mathcal{O}(\epsilon^{-2})$.

In a later work, Xu et al. (2019) improved (7.29), removing the $\mathcal{O}(B^{-1})$ additive term, and were able to prove a $\mathcal{O}(\epsilon^{-5/3})$ rate for SVRPG. This result is necessary to say that SVRPG is indeed better than REINFORCE. For this reason, we report their proof here instead of our original proof for (7.29).

The key contribution by Xu et al. (2019) is the following upper bound on the variance of importance weights, which allows to amortize its effect on the total variance of the estimator. The proof relies on some concepts we introduced in Chapter 6:

Lemma 7.2 (Lemma 6.1 by Xu et al. (2019)) *Under Assumption 7.1, the variance of importance weights is bounded, for all $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta$ and $\tau \in \operatorname{supp}(p_{\boldsymbol{\theta}'})$, as:*

$$\mathbb{V}ar\left[w_{\boldsymbol{\theta}/\boldsymbol{\theta}'}(\tau)\right] \leqslant L_{\mathbb{V}} \left\|\boldsymbol{\theta} - \boldsymbol{\theta}'\right\|^2,$$

where $L_{\mathbb{V}} = 2H\left(2HG_1^2 + G_2\right)(\mathcal{V}_2 + 1)$.

Proof From Equation 6.4, $\mathbb{V}\mathrm{ar}[w_{\boldsymbol{\theta}/\boldsymbol{\theta}'}(\tau)] = d_2(p_{\boldsymbol{\theta}} \| p_{\boldsymbol{\theta}'}) - 1$. Let $d_2(\boldsymbol{\theta}\|\boldsymbol{\theta}')$ be short for $d_2(p_{\boldsymbol{\theta}}\|p_{\boldsymbol{\theta}'})$, the exponentiated Rényi divergence.

The gradient of the divergence w.r.t. the target parameters is:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} d_2(\boldsymbol{\theta}\|\boldsymbol{\theta}') &= \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}'}(\tau) w_{\boldsymbol{\theta}/\boldsymbol{\theta}'}(\tau)^2 \,\mathrm{d}\tau \\
&= \nabla_{\boldsymbol{\theta}} \int \frac{p_{\boldsymbol{\theta}}(\tau)^2}{p_{\boldsymbol{\theta}'}(\tau)} \,\mathrm{d}\tau \\
&= 2 \int \frac{p_{\boldsymbol{\theta}}(\tau)}{p_{\boldsymbol{\theta}'}(\tau)} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) \,\mathrm{d}\tau \\
&= 2 \int \frac{p_{\boldsymbol{\theta}}(\tau)^2}{p_{\boldsymbol{\theta}'}(\tau)} \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \,\mathrm{d}\tau. \qquad (7.30)
\end{aligned}$$

From (7.30) and (3.7) we can see that $\nabla_{\boldsymbol{\theta}} d_2(\boldsymbol{\theta}\|\boldsymbol{\theta}')|_{\boldsymbol{\theta}=\boldsymbol{\theta}'} = 0$. From the mean value theorem:

$$\begin{aligned}
d_2(\boldsymbol{\theta}\|\boldsymbol{\theta}') &= 1 + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}')^T \nabla^2_{\boldsymbol{\theta}} d_2(\boldsymbol{\theta}_\alpha\|\boldsymbol{\theta}')(\boldsymbol{\theta} - \boldsymbol{\theta}') \\
&\leqslant 1 + \frac{1}{2} \left\|\nabla_{\boldsymbol{\theta}} d_2(\boldsymbol{\theta}_\alpha\|\boldsymbol{\theta}')\right\| \left\|\boldsymbol{\theta} - \boldsymbol{\theta}'\right\|^2, \qquad (7.31)
\end{aligned}$$

where $\boldsymbol{\theta}_\alpha = \alpha\boldsymbol{\theta} + (1-\alpha)\boldsymbol{\theta}'$, for some $\alpha \in [0,1]$. We compute the Hessian by differentiating (7.30) once again:

$$\begin{aligned}
\nabla^2_{\boldsymbol{\theta}} d_2(\boldsymbol{\theta}\|\boldsymbol{\theta}') &= 2\nabla_{\boldsymbol{\theta}} \int \frac{p_{\boldsymbol{\theta}}(\tau)^2}{p_{\boldsymbol{\theta}'}(\tau)} \nabla^T_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \,\mathrm{d}\tau \\
&= 2 \int p_{\boldsymbol{\theta}'}(\tau) w_{\boldsymbol{\theta}/\boldsymbol{\theta}'}(\tau)^2 \left(2\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \nabla^T_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) + \nabla^2_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)\right) \mathrm{d}\tau,
\end{aligned}$$

whose spectral norm can be bounded as:

$$\begin{aligned}\left\|\nabla^2_{\boldsymbol{\theta}} d_2(\boldsymbol{\theta}\|\boldsymbol{\theta}')\right\| &\leqslant 2\int p_{\boldsymbol{\theta}'}(\tau) w_{\boldsymbol{\theta}/\boldsymbol{\theta}'}(\tau)^2 \left(2\left\|\nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\tau)\right\|^2 + \left\|\nabla^2_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\tau)\right\|\right) \mathrm{d}\tau \\ &\leqslant 2\left(2H^2G_1^2 + HG_2\right)\left(\mathbb{V}\mathrm{ar}\left[w_{\boldsymbol{\theta}/\boldsymbol{\theta}'}\right] + 1\right) \qquad (7.32) \\ &\leqslant 2H\left(2HG_1^2 + G_2\right)(\mathcal{V}_2 + 1), \qquad (7.33)\end{aligned}$$

where (7.32) is from Assumption 7.2 (2.) and (3.75), and the last inequality is from Assumption 7.2 (4.). Plugging (7.33) into (7.31) gives the result. ■

We now have all the elements required to prove the $\mathcal{O}(\epsilon^{-5/3})$ convergence rate of SVRPG. The proof by (Xu et al., 2019), reported here in our own notation, follows the structure of the proof by Papini et al. (2018, Theorem 4.4), which in turn is heavily inspired by the proof by Reddi et al. (2016) on nonconvex SVRG.

We extend the notation used in the analysis of REINFORCE. Let $\widetilde{\mathcal{D}}^s$ denote the batch collected at the beginning of the s-th epoch, and $\mathcal{D}^s_k$ the mini-batch collected at the k-th iteration of the s-th epoch. Let $\mathcal{F}^s_k$ be the σ-algebra obtained from *all* the data collected *prior* to the k-th iteration of the s-th epoch. We use the following convention: $\mathcal{F}^{s-1}_m$ does not include the batch $\widetilde{\mathcal{D}}^s$, while $\mathcal{F}^s_0$ does. The iterate $\boldsymbol{\theta}^s_k$ is $\mathcal{F}^s_k$-measurable, and $\widetilde{\boldsymbol{\theta}}^s$ is $\mathcal{F}^{s-1}_m$-measurable. We use $\mathbb{E}^s$ to abbreviate $\mathbb{E}_{\widetilde{\mathcal{D}}^s\sim p_{\widetilde{\boldsymbol{\theta}}^s}}\left[\cdot|\mathcal{F}^{s-1}_m\right]$, and $\mathbb{E}^s_k = \mathbb{E}^s[\mathbb{E}_{\mathcal{D}^s_k\sim p_{\boldsymbol{\theta}^s_k}}[\cdot|\mathcal{F}^s_k]]$, which captures the randomness of the latest batch *and* the latest mini-batch. Nested expectations are defined as usual and a simple $\mathbb{E}$ is used to denote expectation over all sources of randomness.

Theorem 7.2 (Theorem 5.5 by Xu et al. (2019)) *Under Assumption 7.2, Algorithm 9 with constant step size $\alpha \leqslant \frac{1}{4L}$, and epoch length m and mini-batch size B satisfying:*

$$\frac{B}{m^2} \geqslant \frac{3(L_{\mathbb{V}}C_g^2 + L_g^2)}{2L^2}, \qquad (7.34)$$

outputs parameters $\boldsymbol{\theta}_{OUT}$ such that:

$$\mathbb{E}\left[\left\|\nabla J(\boldsymbol{\theta}_{OUT})\right\|^2\right] \leqslant \frac{8(J(\boldsymbol{\theta}^*) - J(\boldsymbol{\theta}_0))}{Sm\alpha} + \frac{6\mathcal{V}_1}{N}, \qquad (7.35)$$

where N is the batch size, S the number of epochs, $L_{\mathbb{V}}$ is from Lemma 7.2 and all the other constants are from Lemma 7.1.

Proof Let $v^s_k = \blacktriangledown J(\boldsymbol{\theta}^s_k)$ denote the SVRPG gradient estimate. From Lemma 7.1,

the performance is L-smooth. Hence, from lemma A.2:

$$\begin{aligned}
J(\boldsymbol{\theta}_{k+1}^s) &\geqslant J(\boldsymbol{\theta}_k^s) + \langle \nabla J(\boldsymbol{\theta}_k^s), \boldsymbol{\theta}_{k+1}^s - \boldsymbol{\theta}_k^s \rangle - \frac{L}{2} \left\| \boldsymbol{\theta}_{k+1}^s - \boldsymbol{\theta}_k^s \right\|^2 \\
&= J(\boldsymbol{\theta}_k^s) + \langle \nabla J(\boldsymbol{\theta}_k^s) - v_k^s, \alpha v_k^s \rangle + \alpha \left\| v_k^s \right\|^2 - \frac{L}{2} \left\| \boldsymbol{\theta}_{k+1}^s - \boldsymbol{\theta}_k^s \right\|^2 \\
&\geqslant J(\boldsymbol{\theta}_k^s) - \frac{\alpha}{2} \left\| \nabla J(\boldsymbol{\theta}_k^s) - v_k^s \right\|^2 + \frac{\alpha}{2} \left\| v_k^s \right\|^2 - \frac{L}{2} \left\| \boldsymbol{\theta}_{k+1}^s - \boldsymbol{\theta}_k^s \right\|^2 \qquad (7.36) \\
&\geqslant J(\boldsymbol{\theta}_k^s) - \frac{3\alpha}{4} \left\| \nabla J(\boldsymbol{\theta}_s^k) - v_k^s \right\|^2 + \left(\frac{1}{4\alpha} - \frac{L}{2} \right) \left\| \boldsymbol{\theta}_{k+1}^s - \boldsymbol{\theta}_k^s \right\|^2 \\
&\quad + \frac{\alpha}{8} \left\| \nabla J(\boldsymbol{\theta}_k^s) \right\|^2 \qquad (7.37)
\end{aligned}$$

where (7.36) is from Young's inequality

First, we bound the variance of the SVRPG estimator. Let us use the abbreviation $w_k^s = w_{\widetilde{\boldsymbol{\theta}}^s / \boldsymbol{\theta}_k^s}$:

$$\begin{aligned}
\mathbb{E}_k^s \left[\left\| \nabla J(\boldsymbol{\theta}_k^s) - v_k^s \right\|^2 \right] &= \mathbb{E}_k^s \left[\left\| \nabla J(\boldsymbol{\theta}_k^s) - \widetilde{\mu}^s + \frac{1}{B} \sum_{i=1}^B \left(w_k^s(\tau_i) g(\widetilde{\boldsymbol{\theta}}^s; \tau_i) - g(\boldsymbol{\theta}_k^s; \tau_i) \right) \right\|^2 \right] \\
&\leqslant \mathbb{E}_k^s \left[\left\| \nabla J(\boldsymbol{\theta}_k^s) - \nabla J(\widetilde{\boldsymbol{\theta}}^s) + \frac{1}{B} \sum_{i=1}^B \left(w_k^s(\tau_i) g(\widetilde{\boldsymbol{\theta}}^s; \tau_i) - g(\boldsymbol{\theta}_k^s; \tau_i) \right) \right\|^2 \right] \\
&\quad + \mathbb{E}_k^s \left\| \nabla J(\widetilde{\boldsymbol{\theta}}^s) - \widetilde{\mu}^s \right\|^2 \\
&\leqslant \frac{1}{B^2} \sum_{i=1}^B \mathbb{E}_k^s \left[\left\| \nabla J(\boldsymbol{\theta}_k^s) - \nabla J(\widetilde{\boldsymbol{\theta}}^s) + w_k^s(\tau_i) g(\widetilde{\boldsymbol{\theta}}^s; \tau_i) - g(\boldsymbol{\theta}_k^s; \tau_i) \right\|^2 \right] + \frac{\mathcal{V}_1}{N} \qquad (7.38) \\
&\leqslant \frac{1}{B^2} \sum_{i=1}^B \mathbb{E}_k^s \left[\left\| w_k^s(\tau_i) g(\widetilde{\boldsymbol{\theta}}^s; \tau_i) - g(\boldsymbol{\theta}_k^s; \tau_i) \right\|^2 \right] + \frac{\mathcal{V}_1}{N} \qquad (7.39) \\
&\leqslant \frac{1}{B^2} \sum_{i=1}^B \left(\mathbb{E}_k^s \left[\left\| (w_k^s(\tau_i) - 1) g(\widetilde{\boldsymbol{\theta}}^s; \tau_i) \right\|^2 \right] + \mathbb{E}_k^s \left[\left\| g(\widetilde{\boldsymbol{\theta}}^s; \tau_i) - g(\boldsymbol{\theta}_k^s; \tau_i) \right\|^2 \right] \right) + \frac{\mathcal{V}_1}{N} \\
&\leqslant \frac{1}{B^2} \sum_{i=1}^B \left(C_g^2 \mathbb{V}\mathrm{ar}_k^s [w_k^s(\tau_i)] + L_g^2 \mathbb{E}_k^s \left[\left\| \widetilde{\boldsymbol{\theta}}^s - \boldsymbol{\theta}_k^s \right\|^2 \right] \right) + \frac{\mathcal{V}_1}{N} \qquad (7.40) \\
&\leqslant \frac{C_g^2 L_{\mathbb{V}} + L_g^2}{B} \mathbb{E}_k^s \left[\left\| \widetilde{\boldsymbol{\theta}}^s - \boldsymbol{\theta}_k^s \right\|^2 \right] + \frac{\mathcal{V}_1}{N} \qquad (7.41)
\end{aligned}$$

where we have used the independence of $\widetilde{\mathcal{D}}^s$ and $\mathcal{D}_k^s$, (7.38) is from Lemma 7.1.2, (7.39) is from $\mathbb{E}\left[\left\| \mathbb{E}[\boldsymbol{x}] - \boldsymbol{x} \right\|^2 \right] \leqslant \mathbb{E}\left[\left\| \boldsymbol{x} \right\|^2 \right]$ for any random vector $\boldsymbol{x}$, (7.40) is from Lemma 7.1 (points 3 and 4), and (7.41) is from Lemma 7.2.

From (7.37) and (7.41):

$$\mathbb{E}_k^s\left[J(\boldsymbol{\theta}_{k+1}^s)\right] \geqslant \mathbb{E}_k^s\left[J(\boldsymbol{\theta}_k^s)\right] - \alpha\Psi\mathbb{E}_k^s\left[\left\|\widetilde{\boldsymbol{\theta}}^s - \boldsymbol{\theta}_k^s\right\|^2\right] - \frac{3\alpha\mathcal{V}_1}{4N} + \left(\frac{1}{4\alpha} - \frac{L}{2}\right)\mathbb{E}_k^s\left[\left\|\boldsymbol{\theta}_{k+1}^s - \boldsymbol{\theta}_k^s\right\|^2\right] + \frac{\alpha}{8}\mathbb{E}_k^s\left[\left\|\nabla J(\boldsymbol{\theta}_k^s)\right\|^2\right], \quad (7.42)$$

where $\Psi = 3(C_g^2 L_{\mathbb{V}} + L_g^2)/(4B)$. From Young's inequality in the Peter-Paul variant:

$$\left\|\boldsymbol{\theta}_{k+1}^s - \widetilde{\boldsymbol{\theta}}^s\right\|^2 \leqslant (1+\epsilon)\left\|\boldsymbol{\theta}_{k+1}^s - \boldsymbol{\theta}_k^s\right\|^2 + \left(1 + \frac{1}{\epsilon}\right)\left\|\boldsymbol{\theta}_k^s - \widetilde{\boldsymbol{\theta}}^s\right\|^2, \quad (7.43)$$

for all $\epsilon > 0$. By setting $\epsilon = 2k+1$ and rearranging:

$$\left\|\boldsymbol{\theta}_{k+1}^s - \boldsymbol{\theta}_k^s\right\|^2 \geqslant \frac{1}{2(k+1)}\left\|\boldsymbol{\theta}_{k+1}^s - \widetilde{\boldsymbol{\theta}}^s\right\|^2 - \frac{1}{2k+1}\left\|\boldsymbol{\theta}_k^s - \widetilde{\boldsymbol{\theta}}^s\right\|^2. \quad (7.44)$$

From (7.42) and (7.44), since $\alpha \leqslant \frac{1}{2L}$, after some rearranging:

$$\mathbb{E}_k^s\left[J(\boldsymbol{\theta}_{k+1}^s)\right] - \mathbb{E}_k^s\left[J(\boldsymbol{\theta}_k^s)\right] \geqslant \frac{\alpha}{8}\mathbb{E}_k^s\left[\left\|\nabla J(\boldsymbol{\theta}_k^s)\right\|^2\right] - \frac{3\alpha\mathcal{V}_1}{4N} + \frac{1}{2(k+1)}\left(\frac{1}{4\alpha} - \frac{L}{2}\right)\mathbb{E}_k^s\left[\left\|\boldsymbol{\theta}_{k+1}^s - \widetilde{\boldsymbol{\theta}}^s\right\|^2\right] - \left[\alpha\Psi + \frac{1}{2k+1}\left(\frac{1}{4\alpha} - \frac{L}{2}\right)\right]\mathbb{E}_k^s\left[\left\|\widetilde{\boldsymbol{\theta}}^s - \boldsymbol{\theta}_k^s\right\|^2\right]. \quad (7.45)$$

Summing both sides over $k = 0, 1, \ldots, m-1$, the left-hand side telescopes:

$$\begin{aligned}
\mathbb{E}^s_{0:m-1}\left[J(\widetilde{\boldsymbol{\theta}}^{s+1})\right] - \mathbb{E}^{s-1}_m\left[J(\widetilde{\boldsymbol{\theta}}^{s})\right] &\geqslant \frac{\alpha}{8}\sum_{k=0}^{m-1}\mathbb{E}^s_k\left[\|\nabla J(\boldsymbol{\theta}^s_k)\|^2\right] - \frac{3m\alpha\mathcal{V}_1}{4N} \\
&+ \sum_{k=0}^{m-1}\frac{\frac{1}{2\alpha}-L}{4(k+1)}\mathbb{E}^s_k\left[\left\|\boldsymbol{\theta}^s_{k+1}-\widetilde{\boldsymbol{\theta}}^s\right\|^2\right] - \sum_{k=0}^{m-1}\left[\alpha\Psi + \frac{\frac{1}{2\alpha}-L}{2(2k+1)}\right]\mathbb{E}^s_k\left[\left\|\widetilde{\boldsymbol{\theta}}^s-\boldsymbol{\theta}^s_k\right\|^2\right] \\
&= \frac{\alpha}{8}\sum_{k=0}^{m-1}\mathbb{E}^s_k\left[\|\nabla J(\boldsymbol{\theta}^s_k)\|^2\right] - \frac{3m\alpha\mathcal{V}_1}{4N} + \sum_{k=0}^{m-2}\frac{\frac{1}{2\alpha}-L}{4(k+1)}\mathbb{E}^s_k\left[\left\|\boldsymbol{\theta}^s_{k+1}-\widetilde{\boldsymbol{\theta}}^s\right\|^2\right] \\
&- \sum_{k=1}^{m-1}\left[\alpha\Psi + \frac{\frac{1}{2\alpha}-L}{2(2k+1)}\right]\mathbb{E}^s_k\left[\left\|\widetilde{\boldsymbol{\theta}}^s-\boldsymbol{\theta}^s_k\right\|^2\right] + \frac{\frac{1}{2\alpha}-L}{4m}\mathbb{E}^s_{m-1}\left[\left\|\boldsymbol{\theta}^s_m-\widetilde{\boldsymbol{\theta}}^s\right\|^2\right] \\
&- \left[\alpha\Psi + \frac{1}{4\alpha} - \frac{L}{2}\right]\mathbb{E}^s_k\left[\left\|\widetilde{\boldsymbol{\theta}}^s-\boldsymbol{\theta}^s_0\right\|^2\right] \\
&= \frac{\alpha}{8}\sum_{k=0}^{m-1}\mathbb{E}^s_k\left[\|\nabla J(\boldsymbol{\theta}^s_k)\|^2\right] - \frac{3m\alpha\mathcal{V}_1}{4N} - \sum_{k=1}^{m-1}\left[\frac{\frac{1}{4\alpha}-\frac{L}{2}}{2k(2k+1)} - \alpha\Psi\right]\mathbb{E}^s_k\left[\left\|\widetilde{\boldsymbol{\theta}}^s-\boldsymbol{\theta}^s_k\right\|^2\right] \\
&+ \frac{\frac{1}{2\alpha}-L}{4m}\mathbb{E}^s_{m-1}\left[\left\|\boldsymbol{\theta}^s_m-\widetilde{\boldsymbol{\theta}}^s\right\|^2\right] - \left[\alpha\Psi + \frac{1}{4\alpha} - \frac{L}{2}\right]\mathbb{E}^s_k\left[\left\|\widetilde{\boldsymbol{\theta}}^s-\boldsymbol{\theta}^s_0\right\|^2\right] \\
&= \frac{\alpha}{8}\sum_{k=0}^{m-1}\mathbb{E}^s_k\left[\|\nabla J(\boldsymbol{\theta}^s_k)\|^2\right] - \frac{3m\alpha\mathcal{V}_1}{4N} \\
&- \sum_{k=1}^{m-1}\left[\frac{\frac{1}{4\alpha}-\frac{L}{2}}{2k(2k+1)} - \frac{3\alpha(C_g^2L_{\mathbb{V}}+L_g^2)}{4B}\right]\mathbb{E}^s_k\left[\left\|\widetilde{\boldsymbol{\theta}}^s-\boldsymbol{\theta}^s_k\right\|^2\right] \\
&+ \frac{\frac{1}{2\alpha}-L}{4m}\mathbb{E}^s_{m-1}\left[\left\|\boldsymbol{\theta}^s_m-\widetilde{\boldsymbol{\theta}}^s\right\|^2\right] \\
&- \left[\frac{3\alpha(C_g^2L_{\mathbb{V}}+L_g^2)}{4B} + \frac{1}{4\alpha} - \frac{L}{2}\right]\mathbb{E}^s\left[\left\|\widetilde{\boldsymbol{\theta}}^s-\boldsymbol{\theta}^s_0\right\|^2\right]. \qquad (7.46)
\end{aligned}$$

From the assumptions on α and B/m^2:

$$\mathbb{E}^s_{0:m-1}\left[J(\widetilde{\boldsymbol{\theta}}^{s+1})\right] - \mathbb{E}^{s-1}_m\left[J(\widetilde{\boldsymbol{\theta}}^{s})\right] \geqslant \frac{\alpha}{8}\sum_{k=0}^{m-1}\mathbb{E}^s_k\left[\|\nabla J(\boldsymbol{\theta}^s_k)\|^2\right] - \frac{3m\mathcal{V}_1\alpha}{4N}. \qquad (7.47)$$

Summing both sides over $s = 0, \ldots, S-1$, the left-hand side telescopes:

$$\mathbb{E}[J(\widetilde{\boldsymbol{\theta}}^S)] - J(\boldsymbol{\theta}_0) \geqslant \frac{\alpha}{8}\sum_{s=0}^{S-1}\sum_{k=0}^{m-1}\mathbb{E}^s_k\left[\|\nabla J(\boldsymbol{\theta}^s_k)\|^2\right] - \frac{3Sm\mathcal{V}_1\alpha}{4N}. \qquad (7.48)$$

Rearranging and dividing both sides by the total number of iterations Sm:

$$\frac{1}{Sm}\sum_{s=0}^{S-1}\sum_{k=0}^{m-1}\mathbb{E}^s_k\left[\|\nabla J(\boldsymbol{\theta}^s_k)\|^2\right] \leqslant \frac{8\left(J(\boldsymbol{\theta}^*) - J(\boldsymbol{\theta}_0)\right)}{Sm\alpha} + \frac{6\mathcal{V}_1}{N}. \qquad (7.49)$$

The left-hand side is $\mathbb{E}\left[\|\nabla J(\boldsymbol{\theta}_{\text{OUT}})\|^2\right]$, so the proof is complete. ■

As anticipated, the additive $\mathcal{O}(B^{-1})$ was removed from (7.29). From Theorem 7.2, we can immediately prove the $\mathcal{O}(\epsilon^{-5/3})$ rate:

Corollary 7.3 (Corollary 5.7 by Xu et al. (2019)) *Under Assumption 7.2, for all $\epsilon > 0$, Algorithm 9 with step size $\alpha = 1/(4L)$, batch size $N = \mathcal{O}(\epsilon^{-1})$, mini-batch size $B = \mathcal{O}(\epsilon^{-2/3})$ and epoch length $m = \sqrt{B}$, guarantees $\mathbb{E}\left[\|\nabla J(\boldsymbol{\theta}_{OUT})\|^2\right] \leqslant \epsilon$ after collecting a total of $\mathcal{O}(\epsilon^{-5/3})$ trajectories.*

Proof To obtain $\mathbb{E}\left[\|\nabla J(\boldsymbol{\theta}_{\text{OUT}})\|^2\right] \leqslant \epsilon$ we make sure:

$$\frac{32L\left(J(\boldsymbol{\theta}^*) - J(\boldsymbol{\theta}_0)\right)}{Sm} \leqslant \frac{\epsilon}{2} \qquad \text{and} \qquad \frac{6\mathcal{V}_1}{N} \leqslant \frac{\epsilon}{2},$$

which requires $Sm = \mathcal{O}(\epsilon^{-1})$ and $N = \mathcal{O}(\epsilon^{-1})$. Since $m = \sqrt{B}$, $S = \mathcal{O}(1/(\sqrt{B}\epsilon))$. The total number of trajectories is then:

$$SN + SmB = \mathcal{O}\left(\frac{N}{\sqrt{B}\epsilon} + \frac{B}{\epsilon}\right). \tag{7.50}$$

Finally, by setting $N = \mathcal{O}(\epsilon^{-1})$ and $B = \mathcal{O}(\epsilon^{-2/3})$, we obtain a total of $\mathcal{O}(\epsilon^{-5/3})$ trajectories. ■

The $\mathcal{O}(\epsilon^{-5/3})$ complexity is an $\epsilon^{1/3}$ improvement over REINFORCE, showing that SVRPG is indeed more sample efficient.

7.3.5 Remarks on the SVRPG assumptions

In this section, we show how some of the assumptions made to prove the convergence rate of SVRPG are rather restrictive.

Bounded derivatives. Compared to the smoothing-policy condition used for REINFORCE, Assumption 7.2.2 requires the derivatives of information to be bounded uniformly over the action space rather than in expectation under the policy itself. After we adopted this assumption (Papini et al., 2018, although they go back at least to Sutton et al. (1999)), it was used several times by other authors (Xu et al., 2019; Shen et al., 2019; Xu et al., 2020; Pham et al., 2020; Fallah et al., 2020; Lyu et al., 2020). However, in retrospective, it is quite restrictive. To see why, consider a shallow Gaussian policy with constant standard deviation. From Section 3.3.4, the score is:

$$\nabla \log \pi_{\boldsymbol{\theta}}(a|s) = \frac{a - \mu_{\boldsymbol{\theta}}(s)}{\sigma^2}\phi(s), \tag{7.51}$$

which is unbounded since $\mathcal{A} = \mathbb{R}$ in this case.[9] So this simple and important policy class is smoothing (Section 5.2.1) but does not satisfy Assumption 7.2.2. Unfortunately, removing this strict requirement (e.g., by replacing it with a smoothing condition), is non-trivial. We believe that a result of this kind would represent a valuable step in bridging theory and practice.

Variance of importance weights. We have shown in Chapter 6 that the variance of importance weights is not to be taken lightly. In fact, satisfying Assumption 7.2.4 may require careful policy design or even slight modifications to Algorithm 9. Moreover, the constant $\mathcal{V}_2$ may hide an *exponential* dependency on the task horizon.

As a practically relevant example, consider a shallow Gaussian policy with adaptive standard deviation:

Lemma 7.3 *Given a policy class defined as in Equation 3.29, for all $\boldsymbol{\theta}, \widetilde{\boldsymbol{\theta}} \in \Theta$, the variance of trajectory-wise importance weights is bounded as follows:*

$$\mathbb{V}ar\Big[w_{\widetilde{\boldsymbol{\theta}}/\boldsymbol{\theta}}(\tau)\Big] \leqslant \sup_{s\in\mathcal{S}} \left\{ \frac{\sigma^2_{\boldsymbol{\theta}}}{\sigma_{\widetilde{\boldsymbol{\theta}}}\sigma_2} \exp\left(\frac{(\mu_{\widetilde{\boldsymbol{\theta}}}(s) - \mu_{\boldsymbol{\theta}}(s))^2}{\sigma_2^2} \right) \right\}^H - 1, \tag{7.52}$$

where H is the task horizon and $\sigma_2^2 = 2\sigma^2_{\boldsymbol{\theta}} - \sigma^2_{\widetilde{\boldsymbol{\theta}}}$, provided $\sigma^2_{\boldsymbol{\theta}} > \sigma^2_{\widetilde{\boldsymbol{\theta}}}/2$.

Proof From Equation 6.4 and Lemma 6.4:

$$\mathbb{V}\mathrm{ar}\Big[w_{\widetilde{\boldsymbol{\theta}}/\boldsymbol{\theta}}(\tau)\Big] = d_2(p_{\widetilde{\boldsymbol{\theta}}} \| p_{\boldsymbol{\theta}}) - 1 \leqslant \sup_{s\in\mathcal{S}} \big\{ d_2(\pi_{\widetilde{\boldsymbol{\theta}}}(\cdot|s) \| \pi_{\boldsymbol{\theta}})(\cdot|s) \big\}^H - 1. \tag{7.53}$$

The exponentiated Rényi divergence for the policy class of interest, together with the condition on the standard deviations, can be derived from Equation 6.3, completing the proof. ■

To satisfy Assumption 7.2.4, we first need to ensure that the variance of the current policy is not much smaller than the one of the snapshot policy. Intuitively, the behavioral policy must keep a sufficient amount of exploration to provide fresh data that can be used to evaluate the gradient at a different point (the snapshot). This can be easily achieved by using a fixed standard deviation σ. This could seem quite restrictive, but adaptive variance is problematic for convergence anyway (Section 7.8). In practice, one can monitor the learning process to ensure that the condition $\sigma^2_{\boldsymbol{\theta}} > \sigma^2_{\widetilde{\boldsymbol{\theta}}}/2$ from Lemma 7.3 is satisfied at each inner iteration of Algorithm 9, taking an early snapshot otherwise.

Even with a constant standard deviation, deriving a uniform upper bound $\mathcal{V}_2$ over Θ from Lemma 7.3 may require explicit constraints on the parameters. We are not concerned with an explicit computation of the variance bound given the mostly theoretical interest of the convergence results. Still, this value could be prohibitively

[9] Xu et al. (2020), among others, overcome this issue by assuming $|a| \leqslant a_{\max}$ for some constant $a_{\max}$. However, this means the action distribution is no longer Gaussian. See Section 3.3.4 on bounded-support policies.

large due to the well known exponential dependence of the variance of importance weights on the task horizon (Liu et al., 2018b) already discussed in Chapter 6. This perspective reveals a hidden exponential dependency on the horizon of the sample complexity bounds by Papini et al. (2017); Xu et al. (2019, 2020), which is made explicit in (7.52). A recent algorithm by Shen et al. (2019) achieves the best known sample complexity without relying on this assumption.

7.4 Better Sample Complexity than SVRPG

In the previous section, we have reported a result by Xu et al. (2019) showing that SVRPG enjoys a $\mathcal{O}(\epsilon^{-5/3})$ convergence rate. More recently, Xu et al. (2020) proposed an algorithm called *Stochastic Recursive Variance Reduced Policy Gradient* (SRVR-PG), for which they proved a $\mathcal{O}(\epsilon^{-3/2})$ rate. This is an improvement of $\epsilon^{1/6}$ over SVRPG and a whole $\epsilon^{1/2}$ better than REINFORCE. This algorithm draws inspiration from the SARAH (Nguyen et al., 2017) and SPIDER (Fang et al., 2018) algorithms for finite-sum optimization. It can be seen as a variant of Algorithm 9 where the semi-stochastic gradient is defined as:[10]

$$\blacktriangledown J(\boldsymbol{\theta}_k^s) = \blacktriangledown J(\boldsymbol{\theta}_{k-1}^s) + \widehat{\nabla} J(\boldsymbol{\theta}_k^s; \mathcal{D}) - \frac{1}{B} \sum_{\tau \in \mathcal{D}} w_{\boldsymbol{\theta}_{k-1}^s / \boldsymbol{\theta}_k^s}(\tau) g(\boldsymbol{\theta}_k^s; \tau), \tag{7.54}$$

where $\blacktriangledown J(\boldsymbol{\theta}_0^s) = \widehat{\nabla} J(\widetilde{\boldsymbol{\theta}}^s; \widetilde{\mathcal{D}})$. The recursive definition of $\blacktriangledown J$ gives the name to the algorithm, and is key to its improved sample complexity. Note also how importance weights are always between subsequent policies, not between the current policy and the latest snapshot as in SVRPG. As should be clear from Chapter 6, this reduces the variance introduced by importance weights. Xu et al. (2019) also consider the case of bounded policy spaces and a PGPE-like variant for parameter-based exploration.

Along the same line of work, Shen et al. (2019) proposed the HAPG (Hessian-Aided Policy Gradient) algorithm with $\mathcal{O}(\epsilon^{-3/2})$ sample complexity. In fact, they were the first to prove this result for a policy gradient algorithm. HAPG relies on second order information, but does so in an efficient way. The same result, but without relying on second-order information, is achieved by Lyu et al. (2020), who combine emphatic weights (Sutton et al., 2016) with STORM to improve memory efficiency and off-policy stability. Differently from SRVR-PG, neither of these algorithms employ importance weights, which means they do not require the problematic Assumption 7.2.4. However, they still rely on Assumption 7.2.2.

The $\mathcal{O}(\epsilon^{-3/2})$ rate was believed to be optimal due to a lower bound from smooth nonconvex optimization (Arjevani et al., 2019). A brand-new work by Zhang et al. (2021) challenges this belief by proposing a policy gradient algorithm with $\widetilde{\mathcal{O}}(\epsilon^{-1})$ sample complexity (where $\widetilde{\mathcal{O}}$ hides logarithmic terms). This astounding result is achieved by exploiting the hidden convexity of the problem (Section 3.2.2). Indeed, note that policy optimization is neither a special case (due to the policy-dependent

[10] We report it in the REINFORCE variant for simplicity. Xu et al. (2020) actually suggest to employ the off-policy G(PO)MDP estimator (Equation 3.94).

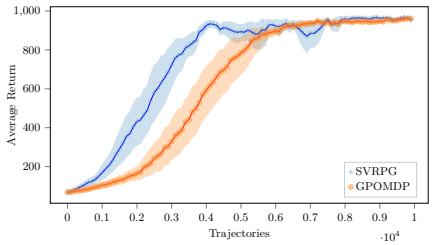


FIGURE 7.1: SVRPG vs G(PO)MDP on Cart-pole, with 90% Student confidence intervals.

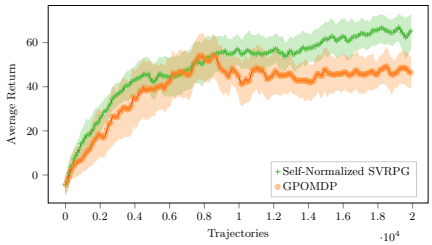


FIGURE 7.2: SVRPG vs G(PO)MDP on Swimmer, with 90% Student confidence intervals.

data distribution) nor a generalization (due to the special structure of the objective function) of the classical stochastic optimization problem. So, we are currently not aware of a lower bound on the sample complexity of policy optimization algorithms.

7.5 Other Related Works

The first application of stochastic variance reduction in RL was to policy evaluation with linear function approximation, which can be formulated as convex-concave saddle problem. Du et al. (2017) adapted SVRG and SAGA to this problem achieving linear convergence rates. Peng et al. (2020b) improved these results with a SCSG-based version. Although more challenging than the original target of SVRG (since it is not strongly convex in the primal variable), this evaluation problem does not have some of main the challenges of policy optimization, namely covariate shift (since the policy is fixed).

The first application of SVRG to policy optimization was by Xu et al. (2017), who heuristically applied the variance reduction technique to TRPO. To match the original setting of SVRG, they turn TRPO into a partially offline algorithm, without addressing the challenges of on-policy optimization we outlined in Section 7.3.2.

Baselines (Section 3.5.4) are still the most used variance-reduction technique in policy gradient algorithms. See (Chung et al., 2020) for a recent overview and novel insights on the topic.

7.6 Empirical Results

In this section, we examine the empirical performance of SVRPG on continuous control problems. All the tasks are from the `rllab` library. See Section C.3 for more details.

According to the theoretical analysis presented in the previous sections, we should observe faster convergence compared to REINFORCE. However, practical implementations of SVRPG must deal with meta-parameter tuning. This is a very complex problem. As often in theoretical convergence guarantees, the step size α prescribed by Theorem 7.2 is too small to be practical, since it is based on

worst-case assumptions. The relationship between α and the batch size N, discussed in Chapter 5, also extends to the mini-batch size B here. Furthermore, the choice B and epoch length m are fundamental to keep the variance of the importance weighted estimates under control. Finally, only an appropriate compromise between N, B, and m can lead to improved sample efficiency in practice.

In the original implementation of SVRPG (Papini et al., 2018)[11], we devised an adaptive meta-parameter schedule based on Adam (Kingma and Ba, 2015). Two separate Adam step-size schedules are used: a global one, $\widetilde{\alpha}^s$, for the "full-gradient updates" (the first of each epoch), and one, α_k^s, for the inner iterations. The epoch length m is also adaptive, as we end the current epoch as soon as the following condition is verified:

$$\frac{\widetilde{\alpha}^s}{N} > \frac{\alpha_k^s}{B}. \tag{7.55}$$

Intuitively, this detects when a full gradient update (i.e., taking the next snapshot) would be more advantageous (in terms of performance improvement per trajectory) than another semi-stochastic gradient update. This heuristic criterion relies on the ability of Adam to implicitly measure the variance of gradient estimates. The batch sizes N and B are still to be selected manually.

In Figure 7.1 we report the performance of SVRPG with $N = 100$ and $B = 10$ on the continuous Cartpole environment with $H = 100$ and $\gamma = 0.99$. The base gradient estimator is G(PO)MDP and the policy is a Gaussian with mean parametrized by a neural network (a single hidden layer of 8 tanh activations) and a separate learned variance parameter. Default hyper-parameters are used for Adam, except the initial step size for $\widetilde{\alpha}^s$ is twice the one for α_k^s to exploit the larger batch size. We compare with G(PO)MDP using default Adam and a batch size of 10. This has the purpose of evaluating the advantage of stabilizing stochastic gradient updates with corrected full-gradients as in SVRPG. Indeed, we can see an improvement in convergence speed, measured by the total number of collected trajectories.

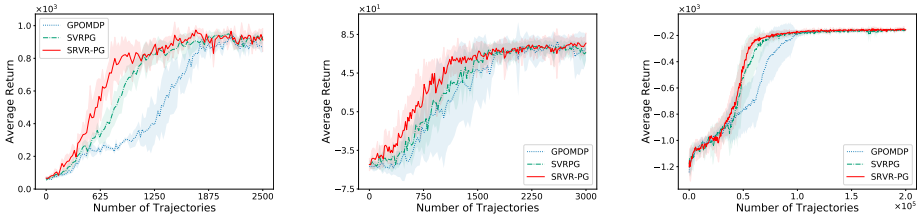
In Figure 7.2 we do the same on the more complex Swimmer task with $H = 500$ and $\gamma = 0.995$. Here we also use self-normalized importance weights (Section 6.1.2) to reduce the variance introduced by importance sampling. The policy mean is parametrized by a 32×32 neural network with tanh activations. Only a slight improvement over G(PO)MDP is observed.

Xu et al. (2020) performed an empirical evaluation of SVRPG and their SRVR-PG algorithm with constant meta-parameters (optimized via grid-search). With the permission of the authors, we report their results on the Cartpole, Mountain Car, and Pendulum tasks in Figure 7.3. The results are coherent with the theoretical ordering of REINFORCE, SVRG, and SRVR-PG.

7.7 Global Convergence Guarantees

Our treatment of convergence guarantees for policy gradient algorithms would not be complete without mentioning the recent breakthroughs on global convergence.

[11] The original code is available at `https://github.com/Dam930/rllab`.



(a) Cartpole (b) Mountain Car (c) Pendulum

FIGURE 7.3: Performance of G(PO)MDP, SVRPG and SRVR-PG on three control tasks (mean $\pm$ standard deviation over 10 random seeds). Reproduced from (Xu et al., 2020, Figure 1) with the permission of the autors.

Although the convergence of policy gradients to local optimal was well known at least since their widespread adoption (Sutton et al., 1999), few works investigated the quality of these optima, likely due to the non-convexity of the problem (Section 3.2.2). In pioneering work, Scherrer and Geist (2014) show that if the policy space Π is a convex set, an ϵ-local optimum of $J(\pi)$ is also close to globally optimal performance. More formally:

$$\begin{aligned} &\frac{\mathrm{d}}{\mathrm{d}\alpha} J((1-\alpha)\pi + \alpha\pi')|_{\alpha=0} \leqslant \epsilon \text{ for all } \pi' \in \Pi \\ &\qquad \implies J(\pi^*) - J(\pi) \leqslant \frac{C^*}{1-\gamma}\left(\frac{\mathcal{E}(\Pi)}{1-\gamma} + \epsilon\right), \end{aligned} \tag{7.56}$$

where $C^* = \left\| d_\mu^{\pi^*} / \mu \right\|_\infty$ is a *concentrability coefficient* accounting for how well the starting state distribution covers the support of optimal state-occupancy, and $\mathcal{E}(\Pi) = \max_{\pi \in \Pi} \min_{\pi' \in \Pi} \{ d_\mu^\pi (T^* V^\pi - T^{\pi'} V^\pi) \}$ measures the irreducible bias introduced by restricting the policy space.[12] Another seminal work in this field is by Neu et al. (2017), who show that an ideal version of TRPO (with unrestricted policy class and exact exact advantage computation) converges to the optimal policy. The key insight is to cast TRPO as approximate mirror descent with the conditional entropy as a mirror map.

Recently, Bhandari and Russo (2019) provided sufficient conditions under which all local optima of the policy optimization problem are actually global optima:

1. Closure of the policy space under policy improvement: for all $\pi \in \Pi_\Theta$, there exists $\pi^+ \in \Pi_\Theta$ such that $T^{\pi^+} V^\pi = T^* V^\pi$;

2. Convex quality function: for all $\pi \in \Pi_\Theta$, the function $\boldsymbol{\theta} \mapsto \mathbb{E}_{s \sim \mu} \left[Q^\pi(s, \pi_{\boldsymbol{\theta}}(s)) \right]$ has no sub-optimal stationary points.

[12] Here, and in the rest of the section, $J(\pi^*) = \max_{\pi \in \Delta_{\mathcal{A}}^{\mathcal{S}}} J(\pi)$ denotes the *unrestricted* global optimum, which may not be attainable within the (possibly restricted) policy class $\Pi \subseteq \Delta_{\mathcal{A}}^{\mathcal{S}}$. Note that this definition of optimality, typical of policy optimization, depends on the starting-state distribution through $J(\pi) = \mathbb{E}_{s \sim \mu} [V^\pi(s)]$. Some authors like Shani et al. (2020), also address the original concept of MDP optimality, where π^* maximizes V^π uniformly over the state space.

They also assume bounded rewards and a full-support starting-state distribution ($\mu(s) > 0$ for all $s \in \mathcal{S}$). Albeit quite strong, these assumptions are satisfied by some interesting problems: directly-parametrized policies (Section 3.3.2) in finite MDPs, tabular Softmax policies (Section 2.6) in finite MDPs with entropy regularization, linear policies for LQR (Section C.1)[13], threshold policies for optimal-stopping problems, and base-stock policies for finite-horizon inventory control (Kunnumkal and Topaloglu, 2008).

Agarwal et al. (2020) were the first to provide explicit convergence rates for policy gradient algorithms to ϵ-optimal policies, that is $J(\pi^*) - J(\pi_{\boldsymbol{\theta}_k}) \leqslant \epsilon$. In particular, they prove that:

- ◇ In finite MDPs under direct policy parametrization, (exact) projected PG converges to an ϵ-optimal policy in $\mathcal{O}(\epsilon^{-2})$ iterations.
- ◇ In finite MDPs, (exact) PG on tabular softmax policies converges asymptotically to an optimal policy. With the addition of entropy regularization, it needs $\mathcal{O}(\epsilon^{-2})$ iterations as well. In the same setting (exact) NPG only needs $\mathcal{O}(\epsilon^{-1})$ iterations.
- ◇ In continuous MDPs with unbounded policy parameters ($\Pi_\Theta \subset \Delta_{\mathcal{A}}^{\mathcal{S}}$ but $\Theta = \mathbb{R}^d$), exact NPG converges in $\mathcal{O}(\epsilon^{-2})$ iterations to a policy that is ϵ-optimal but for an irreducible bias due to restricting the policy class. A similar result can be achieved by NPG with a compatible critic. The latter is the first result for the more realistic case in which the policy gradient is estimated from trajectories.
- ◇ In continuous MDPs with bounded policy parameters ($\Theta \subset \mathbb{R}^d$), exact projected PG converges to an ϵ-optimal policy (but for an irreducible approximation bias) in $\mathcal{O}(\epsilon^{-2})$ iterations.

All of these results depend on concentrability coefficients like C^* from (7.56). According to Agarwal et al. (2020), this indicates that fast policy optimization is fundamentally a matter of efficient exploration.

Shani et al. (2020) prove the convergence of TRPO to ϵ-optimal policies in the finite setting with tabular softmax policies. Once again, it is fundamental to regard TRPO as an approximate mirror descent algorithm. In the exact case, TRPO requires $\mathcal{O}(\epsilon^{-2})$ iterations, which are reduced to $\mathcal{O}(\epsilon^{-1})$ with the addition of entropy regularization. However, the policy obtained in the latter case is optimal w.r.t. a regularized, hence biased objective. In the more realistic sample-based framework, TRPO needs $\mathcal{O}(\epsilon^{-4})$ trajectories to converge to an ϵ-optimal policy, and $\mathcal{O}(\epsilon^{-3})$ with entropy regularization. These sample complexities are not comparable with the one from the previous sections because the notion of convergence is different.

For the same setting (tabular Softmax policies for finite MDPs), Mei et al. (2020) prove a $\mathcal{O}(1/\sqrt{T})$ rate for exact *vanilla* PG *without* entropy regularization, answering to an open question by Agarwal et al. (2020) and matching the best

[13] Global convergence for this setting was previously proved by Fazel et al. (2018) and Yang et al. (2019)

known result for NPG ($\mathcal{O}(\epsilon^{-1})$). They also show that this is the best vanilla PG can achieve. Surprisingly, the addition of entropy regularization allows to obtain a *linear* $\mathcal{O}(e^{-t})$ convergence rate.

Very recently, Zhang et al. (2020a) were able to establish global convergence guarantees for REINFORCE with general parametric policies on finite MDPs. The sample complexity is $\tilde{\mathcal{O}}(\epsilon^{-4})$ in this case.

Also related are works (Cai et al., 2019b; Liu et al., 2019a; Wang et al., 2020) that applies the theory of overparametrized neural networks to prove global convergence of deep reinforcement learning algorithms, and works that study policy optimization as an online-learning problem (Papini et al., 2019a; Rosenberg and Mansour, 2019; Cai et al., 2019a; Jin et al., 2019; Efroni et al., 2020b), establishing sub-linear regret guarantees w.r.t. (globally) optimal performance.

Although most of these theoretical results are limited to specific settings (such as tabular MDPs), they change the perception of what policy optimization algorithms can actually achieve and better explain their empirical success. For most of the history of policy optimization, only convergence to locally optimal policies was thought possible. Future work on PO will have to address the grander challenge of global optimality.

7.8 Concerns on Learning the Policy Variance

In this section, we raise awareness on the possible convergence issues coming from learning the variance of a Gaussian policy via gradient ascent, as is common in deep RL (Duan et al., 2016).

First, we consider the convergence guarantees of REINFORCE. From Theorem 7.1, we can see that the number of trajectories required to converge is proportional to the smoothness constant L of the objective function. We have used Assumption 7.1.2 to ensure L only depends on the policy class regardless of the environment's regularity. If we cast this convergence result to Gaussian policies, the total number of trajectories is proportional to (from Table 5.1):

$$L = \frac{2\phi_{\max} R_{\max}}{\sigma^2(1-\gamma)^2}\left(1 + \frac{2\gamma}{\pi(1-\gamma)}\right). \tag{7.57}$$

This assumes a constant policy standard deviation σ, and makes the sample complexity $\mathcal{O}(\sigma^{-2})$. Unfortunately, this means the convergence guarantee is vacuous when σ is learned as well, especially considering that the typical behavior is convergence to a deterministic policy, i.e., $\sigma \to 0$.

This also applies to the convergence guarantees of PGT and compatible actor critic (Sutton et al., 1999), both in its original version (cf. Equation 3.63) and in the revisited one from Section 7.2.1, which relies on the same smoothness constant (7.57) when applied to Gaussian policies.

It also applies to algorithms relying on Assumption 7.2, if we neglect the other issues raised in Section 7.3.5. Besides SVRPG, this affects the recent works by Xu et al. (2019); Shen et al. (2019); Xu et al. (2020); Pham et al. (2020); Fallah et al.

(2020). As an example, consider the bounding constants provided by Xu et al. (2020) in Appendix D, which are all proportional to σ^{-2}.

This issues are related to the ones discussed for monotonic improvement guarantees in Chapter 5 (variable-variance Gaussian policies are no longer smoothing) and the ones pertaining the variance of importance weights (Chapter 6, Section 7.3.5). It is also related to deterministic PG algorithms (Section 3.9), which informally correspond to the case $\sigma = 0$.

In the next chapter, we propose a way to learn the standard deviation of a Gaussian policy that guarantees monotonic improvement. We believe that studying the convergence behavior of policy gradient algorithms with variable policy variance is another important, overlooked open problem.

Safe Adaptive Stochasticity 8

In this chapter, we study the problem of safe policy optimization when the agent has control over the amount of stochasticity of its own actions. We adopt the *monotonic improvement* framework of Chapter 5, of which this can be considered a direct extension. The novel aspect is that we try not to sacrifice exploration whenever safety requirements leave some margin, since exploratory behavior is fundamental for discovering good policies and for data efficiency. This is a problem of safe exploration (in the sense of Section 4.1). Note, however, that we focus on a limited form of exploration: randomization of actions obtained by *dithering* (i.e., by perturbing the mean action prescribed by the policy). Although this is the most common exploration technique for policy gradient algorithms, exploration in RL is a more profound problem with a vast research tradition (Fruit et al., 2019).

We call this particular aspect of safe exploration *safe adaptive stochasticity* (see Section 4.6 for other, often orthogonal perspectives). We focus on Gaussian policies because they are the most used in practice (Duan et al., 2016) and because they allow to separately parametrize the location and the scale of the state-dependent action distribution. Only scale parameters affect action stochasticity, which allows to better study their effects on safe exploration and to design dedicate update strategies. This is not an easy problem. From the pure safety perspective, we need to treat exploration parameters with care. We have seen in the previous chapters how, even in the simple case of shallow, one-dimensional Gaussian policies, adaptive stochasticity (namely, learning the policy standard deviation σ via gradient ascent like any policy parameter) can disrupt monotonic improvement guarantees (Section 3.3.4), statistical lower bounds of off-policy estimators (Equation 6.3) and even convergence guarantees (Section 7.8). In this chapter, our aim will be to preserve monotonic improvement guarantees under adaptive exploration. As for efficiency, the main challenge resides in correctly weighing the short term costs of random behavior with the long term benefits of exploration. Using the taxonomy of Section 4.1, there is a non-trivial trade-off between active risk and epistemic

risk, since random exploration is a risky tool that the agent uses to increase its knowledge. While the monotonic improvement guarantees are rigorous, our approach to efficiency will be heuristic.[1]

Here is the structure of this last technical chapter. We start in Section 8.1 by generalizing the results from Chapter 5 in several ways: by allowing bounded performance drops, by broadening the concept of safe step size, by considering update directions different from the policy gradient and by extending monotonically improvement guarantees to adaptive-variance Gaussian policies. In Section 8.2, we address the problem of myopic variance learning by introducing a surrogate objective that captures the long-term advantages of exploratory behavior. We present here our heuristic *Meta-Exploring Policy Gradient* (MEPG) algorithm and provide an essential empirical evaluation. In Section 8.3, we develop a safe variant of MEPG with monotonic improvement guarantees, called *Safely Exploring Policy Gradient* (SEPG). This algorithm allows to trade-off safety and exploration, as confirmed by numerical simulations.

This chapter is based on our recent work on safe adaptive stochasticity (Papini et al., 2020).

8.1 Safe Policy Gradient Revisited

In this section, we generalize some of the results on monotonic improvement from Chapter 5 in view of developing safe policy gradient algorithms with adaptive stochasticity.

8.1.1 Bounded-worsening guarantees

First, we relax the monotonic improvement guarantee from Chapter 5 to allow for policy updates with performance differences that are negative, but lower bounded with high probability. Our main purpose is to leave some room for exploratory behavior that may negatively affect performance on the short term in exchange for long-term benefits. However, this generalized requirement is of independent practical interest. Similar requirements, but with a dampening factor on the baseline performance instead of an additive slack, are considered by Wu et al. (2016); Kazerouni et al. (2017) in the bandit literature and recently by Garcelon et al. (2020) for finite MDPs.

The setting is the same as in Chapter 5: given a parametric policy class $\Pi_\Theta = \{\pi_{\boldsymbol{\theta}} \in \Delta_{\mathcal{A}}^{\mathcal{S}} \mid \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^d\}$, we want to design a policy optimization algorithm that guarantees:

$$\mathbb{P}\left(J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant -C_k \mid \boldsymbol{\theta}_k\right) \geqslant 1 - \delta, \tag{8.1}$$

for each iteration $k \geqslant 0$, where δ is a small failure probability and $C_k \geqslant 0$ is a slack representing the maximum allowed performance worsening for the k-th policy

[1] Recent theoretical work by Garcelon et al. (2020) shows that minimum-improvement requirements do not necessarily prevent efficient exploration.

update.[2] The monotonic improvement requirement (5.2) is recovered by setting $C_k = 0$ for all k. However, the latter is a very strict requirement, while positive values of the slack can model interesting practical problems.

Consider, as an example, the *safety w.r.t. a baseline* problem that is typically studied in the setting of batch RL (Section 4.5.1). In this case, the requirement is to never let the performance drop under the one of a baseline policy π_B. In the context of online RL, this could be the initial policy $\pi_{\boldsymbol{\theta}_0}$, which has been carefully engineered to be safe, and lower performances may correspond to unexpected hazards. This safety requirement can be enforced by setting $C_k = J(\boldsymbol{\theta}_k) - J(\boldsymbol{\theta}_0)$ in Condition 8.1.

8.1.2 Less exploitative policy updates

From an exploration perspective, C_k can be regarded as an *exploration budget*: the amount of immediate performance we are willing to pay to invest in exploratory behavior. The criterion used in Chapter 5 to select the step size of policy gradient updates is completely exploitative, as it consists in maximizing immediate performance improvement (Corollary 5.4). We provide here a more general guarantee (we consider exact policy gradient updates for ease of exposition):

Theorem 8.1 *Let Π_Θ be a (ξ_1, ξ_2, ξ_3)-smoothing policy class. Let $\boldsymbol{\theta}_k \in \Theta$ and $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \nabla J(\boldsymbol{\theta}_k)$. The following guarantees Condition 8.1 holds for a slack $C_k \geqslant 0$:*

$$\alpha_k \leqslant \frac{1}{L}\left(1 + \sqrt{1 + \frac{2LC_k}{\|\nabla J(\boldsymbol{\theta}_k)\|^2}}\right), \tag{8.2}$$

where $L = \frac{R_{\max}}{(1-\gamma)^2}\left(\frac{2\gamma\xi_1^2}{1-\gamma} + \xi_2 + \xi_3\right)$.

Proof This follows from Theorem 5.3 by requiring that the lower bound on performance improvement be greater or equal than $-C_k$, and solving for α_k. ■

A possible heuristic, intended to speed up learning, is to use the largest step size α_k that satisfies (8.2). Inequality 8.2 meets the intuition that the larger the allowed performance loss C_k, the larger the step we can dare to take. Note that, by setting $C_k = 0$ (MI), we already get a step size that is double the *greedy* one from Corollary 5.4. Interestingly, the largest step size is not constant like the greedy one and is larger for smaller gradient norms.

8.1.3 Surrogate objectives

In some cases, we would like to optimize an objective different than expected return while satisfying the original improvement requirement (8.1). In this section, we provide a way to do so for smoothing policies.

[2] From Corollary 5.7 we know that we could also require a small positive improvement ($C_k > 0$). However, this would be too small to be of any practical relevance, and would leave even less room for exploration.

Let $\mathcal{L} : \Theta \to \mathbb{R}$ be any differentiable objective function we care to maximize. In the following, let λ_k denote the *scalar projection* of $\nabla J(\boldsymbol{\theta}_k)$ onto $\nabla \mathcal{L}(\boldsymbol{\theta}_k)$:

$$\lambda_k \coloneqq \frac{\langle \nabla J(\boldsymbol{\theta}_k), \nabla \mathcal{L}(\boldsymbol{\theta}_k) \rangle}{\|\nabla \mathcal{L}(\boldsymbol{\theta}_k)\|} \tag{8.3}$$

The following theorem provides a safe update for a smoothing policy in the surrogate gradient direction:

Theorem 8.2 *Let Π_Θ be a (ξ_1, ξ_2, ξ_3)-smoothing policy class, $\boldsymbol{\theta}_k \in \Theta$, and $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \eta_k \nabla \mathcal{L}(\boldsymbol{\theta}_k)$, where $\eta_k \geqslant 0$ is a step size. For any $C_k \geqslant 0$, provided $\lambda_k \neq 0$, the following is enough to guarantee Condition* (8.1)*:*

$$\eta_k \leqslant \frac{|\lambda_k|}{L \|\nabla \mathcal{L}(\boldsymbol{\theta}_k)\|} \left(\operatorname{sign}(\lambda_k) + \sqrt{1 + \frac{2LC_k}{\lambda_k^2}} \right), \tag{8.4}$$

where $L = \frac{R_{\max}}{(1-\gamma)^2} \left(\frac{2\gamma \xi_1^2}{1-\gamma} + \xi_2 + \xi_3 \right)$. *If* $\lambda_k = 0$, *it is enough to select* $\eta_k \leqslant \frac{1}{\|\nabla \mathcal{L}(\boldsymbol{\theta}_k)\|} \sqrt{\frac{2C_k}{L}}$.

Proof Let $\boldsymbol{x}_k \coloneqq \nabla \mathcal{L}(\boldsymbol{\theta}_k)$ for brevity.[3] From Theorem 5.2 with $\Delta \boldsymbol{\theta} = \eta_k \boldsymbol{x}_k$:

$$J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant \eta_k \langle \boldsymbol{x}_k, \nabla J(\boldsymbol{\theta}_k) \rangle - \eta_k^2 \frac{L}{2} \|\boldsymbol{x}_k\|^2 \coloneqq f(\eta_k). \tag{8.5}$$

Intuitively, the more $\boldsymbol{x}_k$ agrees with the improvement direction $\nabla J(\boldsymbol{\theta}_k)$, the more improvement can be guaranteed. We first assume $\langle \boldsymbol{x}_k, \nabla J(\boldsymbol{\theta}_k) \rangle \neq 0$, which implies $\lambda_k \neq 0$. Solving $f(\eta_k) \geqslant -C_k$ for η_k yields:

$$\eta_k \leqslant \frac{|\lambda_k|}{L \|\boldsymbol{x}_k\|} \left(\operatorname{sign}(\lambda_k) + \sqrt{1 + \frac{2LC_k}{\lambda_k^2}} \right),$$

If $C_k = 0$ and $\lambda_k < 0$ (i.e.,, $\boldsymbol{x}_k$ does not agree with the original gradient), the safe step size is zero: there is no way to follow $\boldsymbol{x}$ without reducing the expected return. Instead, if $C_k < 0$ (bounded worsening), a small-enough step in the direction of $\boldsymbol{x}_k$ is always safe.

We now consider the special case $\langle \boldsymbol{x}_k, \nabla J(\boldsymbol{\theta}_k) \rangle = 0$, i.e., $\lambda_k = 0$. The two gradients are orthogonal, so the damage following $\boldsymbol{x}_k$ can do is only due to the curvature. We have $f(\eta_k) = -\eta_k^2 \frac{L}{2} \left\| \boldsymbol{x}_k^2 \right\|$, and solving $f(\eta_k) \geqslant -C_k$ for η_k we get:

$$\eta_k \leqslant \frac{1}{\|\boldsymbol{x}_k\|} \sqrt{\frac{2C_k}{L}}.$$

■

[3] In fact, this theorem holds for an arbitrary update direction, not necessarily corresponding to the gradient of a differentiable objective.

Note that Condition (8.1) can always be guaranteed, even if the improvement direction $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)$ is not explicitly followed. However, if the scalar projection λ_k is negative and we require monotonic improvement ($C_k = 0$), the only safe step size is zero. This corresponds to the case in which maximizing the surrogate objective can only reduce the original one, so no safe update is possible.

8.1.4 Safe updates for Gaussian policies

Let us now focus on scalar-action, shallow Gaussian policies:

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{1}{\sqrt{2\pi}\sigma_{\boldsymbol{\theta}}(s)} \exp\left(-\frac{(a - \mu_{\boldsymbol{\theta}}(s))^2}{2\sigma^2_{\boldsymbol{\theta}}(s)}\right), \tag{8.6}$$

In particular, we consider the following parametrization:

$$\mu_{\boldsymbol{\theta}}(s) = \boldsymbol{\upsilon}^T \phi(s), \qquad \sigma_{\boldsymbol{\theta}} = e^{\omega}, \tag{8.7}$$

where we have divided policy parameters $\boldsymbol{\theta} \in \mathbb{R}^{d+1}$ into mean parameters $\boldsymbol{\upsilon} \in \mathbb{R}^d$ and a scalar variance parameter $\omega \in \mathbb{R}$, and $\phi : \mathcal{S} \to \mathbb{R}^d$ is a feature map such that $\sup_{s\in\mathcal{S}} \|\phi(s)\| \leqslant \phi_{\max}$. By convention $\boldsymbol{\upsilon}$ corresponds to the first d policy parameters and $\omega = \theta_{d+1}$. Note that parametrizing the standard deviation in this way makes the problem unconstrained ($\Theta = \mathbb{R}^{d+1}$) while ensuring $\sigma_{\boldsymbol{\theta}} \geqslant 0$ and is continuous w.r.t. policy parameters for all $\boldsymbol{\theta} \in \Theta$. Deterministic behavior is attained only in the limit $\omega \to -\infty$, so all the policies of Π_Θ are stochastic. We have reduced adaptive exploration to the problem of selecting ω at each iteration. For clarity, we will use $\boldsymbol{\upsilon}$ and ω in place of $\boldsymbol{\theta}$ whenever possible. For instance, we will write $J(\boldsymbol{\upsilon}, \omega)$ in place of $J(\boldsymbol{\theta})$ for the performance.

From Section 5.2.1 we know that this class of policies is smooth w.r.t. mean parameters $\boldsymbol{\upsilon}$ if we fix the standard deviation, i.e., $\omega = \text{const}$. Let us restate the step size condition from Theorem 8.1 for this special case, highlighting the role of the standard deviation $\sigma_\omega = e^\omega = \text{const}$:

$$\alpha_k \leqslant \frac{\sigma^2_\omega}{F}\left(1 + \sqrt{1 + \frac{2FC_k}{\sigma^2_\omega \|\nabla_{\boldsymbol{\upsilon}} J(\boldsymbol{\upsilon}_k, \omega)\|^2}}\right), \tag{8.8}$$

where $F = \frac{2\phi^2_{\max} R_{\max}}{(1-\gamma)^2}\left(1 + \frac{2\gamma}{\pi(1-\gamma)}\right)$ will be a recurring constant in this chapter. As already observed in the previous chapters, a larger standard deviation makes the objective function smoother and allows to take larger gradient steps. This is coherent with the intuitive and empirical arguments by Ahmed et al. (2019) on the effects of entropy on policy optimization.

Before thinking about how we can adaptively select ω to improve the learning speed, we must verify it can be modified online in a safe way. Unfortunately, as discussed in Section 5.2.1, Gaussian policies are no longer smoothing, hence loose the improvement guarantees, when we also learn the standard deviation. A workaround would be to replace σ_ω with a lower bound, which can be imposed by constraining

the parameter space Ω or by changing the parametrization. However, this would make the improvement bounds unnecessarily conservative, and would prevent the agent to converge to deterministic behavior in the end. However, the next lemma shows that the variance parameter can be safely updated *when the mean parameters are kept fixed*:

Lemma 8.1 *Consider the class of Gaussian policies parametrized as in* (8.7)*, but with fixed mean parameters $\boldsymbol{\upsilon} = \text{const}$. This class of policies is $\left(\frac{4}{\sqrt{2\pi e}}, 2, 2\right)$-smoothing.*

Proof Since the mean parameters $\boldsymbol{\upsilon}$ are fixed, the policy parameter space can be restricted to $\mathbb{R}$. Recall that $\sigma_\omega = e^\omega$. We need the following derivatives:

$$\begin{aligned}\nabla_\omega \log \pi_{\boldsymbol{\upsilon},\omega}(a|s) &= \nabla_\omega \left(-\omega - \frac{1}{2} e^{-2\omega}(a - \mu_{\boldsymbol{\upsilon}}(s))^2\right) \\ &= e^{-2\omega}(a - \mu_{\boldsymbol{\upsilon}}(s))^2 - 1, \end{aligned} \tag{8.9}$$

$$\nabla_\omega^2 \log \pi_{\boldsymbol{\upsilon},\omega}(a|s) = -2e^{-2\omega}(a - \mu_{\boldsymbol{\upsilon}}(s))^2. \tag{8.10}$$

Let $x \coloneqq e^{-\omega}(a - \mu_{\boldsymbol{\upsilon}}(s))$, and note that $\nabla_a x = e^{-\omega}$. First we compute ξ_1:

$$\begin{aligned}&\sup_{s\in\mathcal{S}} \mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\upsilon},\omega}} \left[\|\nabla_\omega \log \pi_{\boldsymbol{\upsilon},\omega}(a|s)\|\right] \\ &\quad= \sup_{s\in\mathcal{S}} \frac{e^{-\omega}}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{-\frac{1}{2}e^{-2\omega}(a-\mu_{\boldsymbol{\upsilon}}(s))^2} \left|e^{-2\omega}(a - \mu_{\boldsymbol{\upsilon}}(s))^2 - 1\right| \mathrm{d}a \\ &\quad= \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{-x^2/2} |x^2 - 1| \,\mathrm{d}x \\ &\quad= \frac{4}{\sqrt{2\pi e}} \coloneqq \xi_1. \end{aligned} \tag{8.11}$$

Next, we compute ξ_2:

$$\begin{aligned}&\sup_{s\in\mathcal{S}} \mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\upsilon},\omega}} \left[\|\nabla_\omega \log \pi_{\boldsymbol{\upsilon},\omega}(a|s)\|^2\right] \\ &\quad= \sup_{s\in\mathcal{S}} \frac{e^{-\omega}}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{-\frac{1}{2}e^{-2\omega}(a-\mu_{\boldsymbol{\upsilon}}(s))^2} \left(e^{-2\omega}(a - \mu_{\boldsymbol{\upsilon}}(s))^2 - 1\right)^2 \mathrm{d}a \\ &\quad= \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{-x^2/2}(x^2 - 1)^2 \,\mathrm{d}x = 2 \coloneqq \xi_2. \end{aligned} \tag{8.12}$$

Finally, we compute ξ_3:

$$\begin{aligned}&\sup_{s\in\mathcal{S}} \mathop{\mathbb{E}}_{a\sim\pi_{\boldsymbol{\upsilon},\omega}} \left[\left\|\nabla_\omega^2 \log \pi_{\boldsymbol{\upsilon},\omega}(a|s)\right\|\right] \\ &\quad= \sup_{s\in\mathcal{S}} \frac{e^{-\omega}}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{-\frac{1}{2}e^{-2\omega}(a-\mu_{\boldsymbol{\upsilon}}(s))^2} \left|-2e^{-2\omega}(a - \mu_{\boldsymbol{\upsilon}}(s))^2\right| \mathrm{d}a \\ &\quad= \frac{2}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{-x^2/2} x^2 \,\mathrm{d}x = 2 \coloneqq \xi_3. \end{aligned} \tag{8.13}$$

Indeed, the computed constants are independent from the value of ω. ■

Hence, if we keep $\boldsymbol{\upsilon}$ fixed, we can update the variance parameter as $\omega_{k+1} = \beta_k \nabla_\omega J(\boldsymbol{\upsilon}, \omega_k)$, where $\beta_k \geqslant 0$ is a step size. From Theorem 8.1, the following is enough to guarantee a performance worsening of at most C_k:

$$\beta_k \leqslant \frac{1}{G}\left(1 + \sqrt{1 + \frac{2GC_k}{\|\nabla_\omega J(\boldsymbol{\upsilon}, \omega_k)\|^2}}\right), \tag{8.14}$$

where $G = \frac{4R_{\max}}{(1-\gamma)^2}\left(1 + \frac{4\gamma}{\pi e(1-\gamma)}\right)$ will be a recurring constant in this chapter.

A possible way to update both mean and variance parameters is to do so in an *alternating* fashion. By first updating $\boldsymbol{\upsilon}$ (while keeping ω fixed) with the largest step size α satisfying (8.2), then updating ω (while keeping $\boldsymbol{\upsilon}$ fixed) with the largest step size β satisfying (8.14), we can guarantee Condition 8.1 holds at each step. As we will remark in Section 8.3, in practice each update requires fresh on-policy data to estimate the policy gradient, once with respect to $\boldsymbol{\upsilon}$, once with respect to ω under the updated mean parameters, and so on.

8.2 Adaptive Stochasticity

Learning the variance parameters of Gaussian policies via policy gradient is a common practice (Duan et al., 2016), and we have shown in the previous section that it is also compatible with improvement guarantees under an alternating update scheme. However, this approach does not capture the long-term benefits of stochastic exploration. In this section, we propose a more far-sighted objective for exploration parameters. Although our choices are guided by the theory of safe policy gradients, the resulting *Meta-Exploring Policy Gradient* (MEPG) algorithm is entirely heuristic and has no safety guarantees: we leave to Section 8.3 the task of devising policy updates that are both safe and far-sighted. However, MEPG could be of independent interest.

8.2.1 Meta-Exploring Policy Gradient

Consider the Gaussian policy defined in Section 8.1.4. The *naïve* policy-gradient update for the variance parameter ω is:

$$\omega_{k+1} \leftarrow \omega_k + \beta_k \nabla_\omega J(\boldsymbol{\upsilon}_k, \omega_k). \tag{8.15}$$

However, the effects of σ on the optimization landscape, exposed by equation 8.8, suggest to treat it with particular care, both to exploit its potential and to avoid its risks. In fact, adjusting the policy variance with policy gradient tends to degenerate too early into quasi-deterministic policies, getting stuck in local optima or even causing divergence issues (see Section 7.8). Entropy regularization (Section 3.11) is a way to overcome this problem by favoring more stochastic policies. We propose an alternative solution inspired by results on safe policy gradients.

Algorithm 10 MEPG (Meta-Exploring Policy Gradient)

Input: Initial parameters $\boldsymbol{v}_0$ and ω_0, step size $\alpha > 0$, meta step size $\eta > 0$, batch size N
for $k = 1, 2, \ldots$ **do**
  Collect a batch of N trajectories with $\pi_{\boldsymbol{v}_k, \omega_k}$
  Estimate $\widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k)$, $\widehat{\nabla}_{\omega} J(\boldsymbol{v}_k, \omega_k)$ and $\widehat{\nabla}_{\omega} \left\| \nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) \right\|$
  $\widehat{\nabla}_{\omega} \mathcal{L}(\boldsymbol{v}_k, \omega_k) = \widehat{\nabla}_{\omega} J(\boldsymbol{v}_k, \omega_k) + \alpha e^{2\omega_k} \left(2 \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) \right\| + \widehat{\nabla}_{\omega} \left\| \nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) \right\| \right)$
  $\omega_{k+1} \leftarrow \omega_k + \eta \widehat{\nabla}_{\omega} \mathcal{L}(\boldsymbol{v}_k, \omega_k) / \left\| \widehat{\nabla}_{\omega} \mathcal{L}(\boldsymbol{v}_k, \omega_k) \right\|$
  $\boldsymbol{v}_{k+1} \leftarrow \boldsymbol{v}_k + \alpha e^{2\omega_k} \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) / \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) \right\|$
end for

We modify actor-only PG (Algorithm 2) in two ways. First, we make the step size *for the mean parameters* α dependent on the policy variance, like the safe step size from (8.8). In particular, we use the following:

$$\alpha_k = \frac{\alpha \sigma_{\omega_k}^2}{\left\| \nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) \right\|}, \tag{8.16}$$

where $\alpha > 0$ is a hyper-parameter. This has both the effect of reducing the step size when a small σ makes the optimization landscape less smooth, preventing oscillations, and increasing it when a large σ allows it to do so, increasing the learning speed.[4] We also divide the step size by the norm of the gradient. This is a common normalization technique (Peters and Schaal, 2008a), and is also motivated by Corollary 5.6 on safe stochastic gradient updates.

As for the variance parameter ω, we treat it as a separate *meta-parameter* and we learn it in a meta-gradient fashion (Sutton, 1992; Schraudolph, 1999; Veeriah et al., 2017; Xu et al., 2018). Specifically, we employ a more far-sighted learning objective to avoid premature convergence to deterministic behavior. To do so, we look at the target performance *one step in the future*:

$$\begin{aligned} & J\left(\boldsymbol{v}_k + \alpha \sigma_{\omega_k}^2 \frac{\nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k)}{\left\| \nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) \right\|}, \, \omega_k \right) \\ & \simeq J(\boldsymbol{v}_k, \omega_k) + \alpha \sigma_{\omega_k}^2 \left\| \nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) \right\| \coloneqq \mathcal{L}(\boldsymbol{v}_k, \omega_k), \end{aligned} \tag{8.17}$$

where we performed a first-order approximation. The gradient of $\mathcal{L}$ w.r.t. ω is:

$$\begin{aligned} \nabla_{\omega} \mathcal{L}(\boldsymbol{v}_k, \omega_k) = \nabla_{\omega} J(\boldsymbol{v}_k, \omega_k) & + 2\alpha \sigma_{\omega_k}^2 \left\| \nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) \right\| \\ & + \alpha \sigma_{\omega_k}^2 \nabla_{\omega} \left\| \nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) \right\|. \end{aligned} \tag{8.18}$$

We provide an intuitive interpretation of this update direction. The first term of the sum is the usual policy gradient w.r.t. ω, and accounts for the immediate effect

[4] This is exactly what a natural gradient update does in a pure Gaussian setting (Miyamae et al., 2010), since σ^2 is the inverse Fisher information w.r.t. the mean parameters of a Gaussian distribution (Equation 3.119).

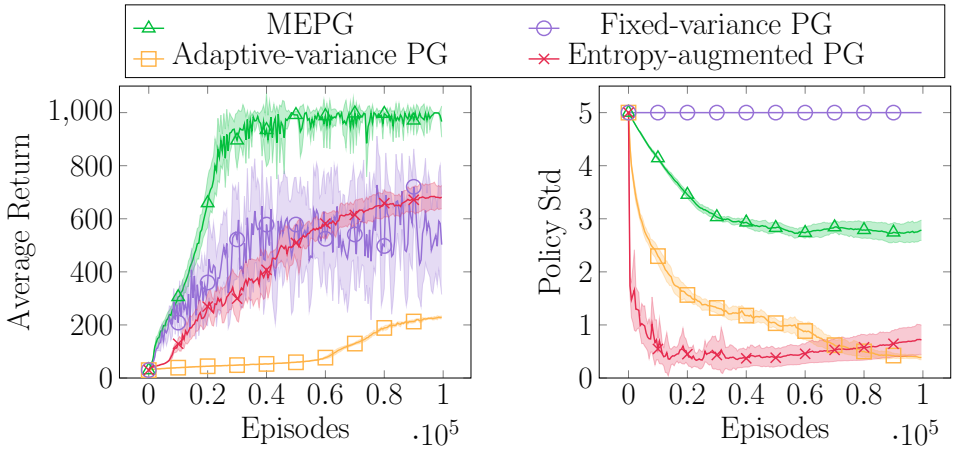


FIGURE 8.1: Average return (undiscounted) and policy standard deviation per episode of MEPG, fixed-variance PG, adaptive-variance PG and entropy-augmented PG on the continuous Cart-Pole task, starting from $\sigma = 5$; averaged over 10 independent runs with 95% Student's t-confidence intervals.

of policy stochasticity on performance. The intuitive role of the second term is to increase the step size α_k, more so if the gradient w.r.t. $\boldsymbol{\upsilon}$ is large. The third term tries to modify the policy variance to increase the gradient norm and can be seen as a way to escape local optima. The last two terms, together, account for the long-term effects of modifying the policy variance. We propose to update ω in the direction of the (normalized) meta-gradient $\nabla_\omega \mathcal{L}$ using a meta-step size $\eta > 0$:

$$\omega_{k+1} \leftarrow \omega_k + \eta \frac{\nabla_\omega \mathcal{L}(\boldsymbol{\upsilon}_k, \omega_k)}{\|\nabla_\omega \mathcal{L}(\boldsymbol{\upsilon}_k, \omega_k)\|}. \tag{8.19}$$

In practice, exact gradients are not available. The policy gradient for the mean-parameter update can be estimated with G(PO)MDP (3.77), and the same is true for $\nabla_\omega J$ in the definition of the meta-gradient. Computing $\widehat{\nabla}_\omega \mathcal{L}$ also requires estimating the mixed-derivative term $\nabla_\omega \|\nabla_{\boldsymbol{\upsilon}} J(\boldsymbol{\upsilon}_k, \omega_k)\|$, which is computationally no more expensive, but could suffer from more variance. See Papini et al. (2020, Appendix B) for an unbiased estimator. The pseudocode for the resulting algorithm, called *Meta-Exploring Policy Gradient* (MEPG), is provided in Algorithm 10. We remark once again that this is a heuristic algorithm with no safety guarantees. In the next section, we provide an empirical evaluation. Developing a safe version of MEPG is the goal of Section 8.3.

8.2.2 Empirical evaluation of MEPG

We test MEPG on continuous-action Cart-Pole (Section C.2) with an horizon $H = 1000$. The long horizon makes the task quite challenging (as a comparison, note that SVRPG was tested on $H = 100$). An agent that does not explore enough

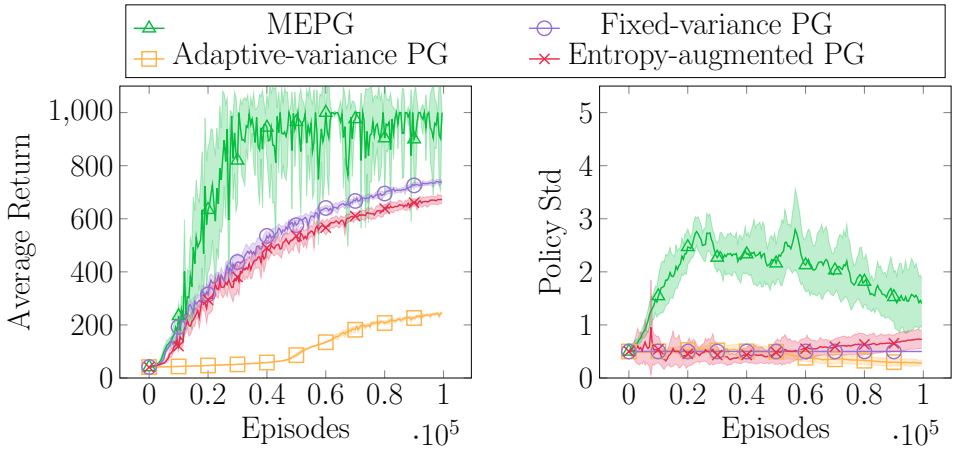


Figure 8.2: Average return (undiscounted) and policy standard deviation per episode of MEPG, fixed-variance PG, adaptive-variance PG and entropy-augmented PG on the continuous Cart-Pole task, starting from $\sigma = 0.5$; averaged over 10 independent runs with 95% Student's t-confidence intervals.

can easily get stuck in suboptimal optima, for instance by focusing on keeping the pole upright without caring to keep the cart within the allowed region as long as possible. Figures 8.1 and 8.2 show the performance (1000 is the maximum) and the policy standard deviation of MEPG and three versions of PG (with the G(PO)MDP gradient estimator). In fixed-variance PG, the policy variance parameter is kept constant. In adaptive-variance PG, it is learned via gradient ascent as any other parameter, using the naïve update (8.15). Entropy-augmented PG is the same, but with entropy regularization (Section 3.11). For each algorithm, the best hyper-parameters (step sizes and entropy coefficient τ) have been selected by grid search using 5 separate random seeds. Two very different initializations are considered for the standard deviation: $\sigma_{\omega_0} = 5$ (Figure 8.1) and $\sigma_{\omega_0} = 0.5$ (Figure 8.2). As shown by the behavior of fixed-variance G(PO)MDP, the former constant value is too large to achieve optimal performance at convergence, while the latter is too small to properly explore the environment. As expected, adaptive-variance G(PO)MDP is too greedy and ends up always reducing the standard deviation. Besides preventing exploration, divergence issues force us to use a smaller step size ($\alpha = 0.01$ instead of 0.1), resulting in slower learning. This problem is fixed by the entropy bonus, which prevents the policy from becoming deterministic and allows to use the larger learning rate ($\alpha = 0.1$). However, entropy-augmented PG does not perform significantly better than its fixed-variance counterpart on this task, as the amount of exploration needed to find the global optimum is not maintained (or is pursued too late). Instead, MEPG is able to settle on an intermediate value with both variance initializations. This allows both to learn faster and to achieve optimal performance, although non-negligible oscillations can be observed. This oscillations are partly due to the variance of the meta-gradient estimator, and can

be mitigated by the conservative updates we propose in the next section. Table 8.1 provides a recap of the selected meta-parameters.

8.3 Safely Exploring Policy Gradient

In this section, we use the results on safe gradient updates of Gaussian policies with adaptive variance to design a variant of Algorithm 10 with improvement guarantees.

8.3.1 Exact framework

Let us first consider the FO setting for simplicity, assuming access to exact policy gradients. We have shown in Section 8.1.4 that alternately updating the mean parameters with a step size α_k satisfying (8.8), and the variance parameter with a step size β_k satisfying (8.14), ensures $J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant -C_k$ for all k.

However, this result still pertains naïve variance updates, which suffer from all the problems discussed in Section 8.2. The next question is how to optimize the surrogate exploratory objective $\mathcal{L}$ from (8.17) while satisfying the original constraint (8.1) on the expected return. Luckily, we can use a step size from Theorem 8.2 to safely replace $\nabla_\omega J$ with the meta gradient $\nabla_\omega \mathcal{L}$ from (8.18) in the variance update:

$$\omega_{k+1} \leftarrow \omega_k + \eta_k \nabla_\omega \mathcal{L}(\boldsymbol{\upsilon}_k, \omega_k). \tag{8.20}$$

From Lemma 8.1 and Theorem 8.2, the following is enough to guarantee Condition 8.1:

$$\eta_k \leqslant \frac{|\lambda_k|}{G \|\nabla_\omega \mathcal{L}(\boldsymbol{\upsilon}_k, \omega_k)\|} \left(\operatorname{sign}(\lambda_k) + \sqrt{1 + \frac{2GC_k}{\lambda_k^2}} \right), \tag{8.21}$$

where λ_k is the scalar projection (8.3) of $\nabla_\omega J(\boldsymbol{\upsilon}_k, \omega_k)$ onto $\nabla_\omega \mathcal{L}(\boldsymbol{\upsilon}_k, \omega_k)$.

In particular, with the idea of maximizing learning speed, we employ the largest step size $\overline{\alpha}_k$ satisfying (8.2) for mean updates and the largest step size $\overline{\eta}_k$ satisfying (8.21) for variance updates. We perform these updates in an alternate fashion as discussed in Section 8.1.4:

$$\begin{aligned} \boldsymbol{\upsilon}_{k+1} &\leftarrow \boldsymbol{\upsilon}_k + \overline{\alpha}_k \nabla_{\boldsymbol{\upsilon}} J(\boldsymbol{\upsilon}_k, \omega_k), \\ \omega_{k+2} &\leftarrow \omega_{k+1} + \overline{\eta}_{k+1} \nabla_\omega \mathcal{L}(\boldsymbol{\upsilon}_{k+1}, \omega_{k+1}), \end{aligned}$$

where $\omega_{k+1} \equiv \omega_k$ and $\boldsymbol{\upsilon}_{k+2} \equiv \boldsymbol{\upsilon}_{k+1}$. This guarantees bounded performance worsening (8.1) at each iteration k. We can indeed interpret the slack C_k as an exploration budget since, the larger C_k, the larger a step is performed in the direction of increasing *exploratory* objective.

We call the resulting algorithm SEPG. We postpone the pseudocode to the next section, where we consider the more realistic FSO setting.

Algorithm	α	η	τ
MEPG ($\sigma_{\omega_0} = 5.0$)	0.1	0.01	
MEPG ($\sigma_{\omega_0} = 0.5$)	1.0	0.1	
Fixed-variance PG ($\sigma_{\omega_0} = 5.0$)	1.0		
Fixed-variance PG ($\sigma_{\omega_0} = 0.5$)	0.1		
Adaptive-variance PG	0.01		
Entropy-augmented PG	0.1		0.1

Table 8.1: Meta-parameters for the Cart-Pole experiment, optimized via grid search. When not specified, the same value was selected for the two initializations.

8.3.2 Approximate framework

In this section, we show how to adapt the safe step sizes from Section 8.3.1 to take gradient estimation errors into account.

Let $\widehat{\nabla}_{\boldsymbol{v}} J$, $\widehat{\nabla}_{\omega} J$ and $\widehat{\nabla}_{\omega} \mathcal{L}$ be unbiased estimators of $\nabla_{\boldsymbol{v}} J$, $\nabla_{\omega} J$ and $\nabla_{\omega} \mathcal{L}$, respectively, each using a batch of N trajectories. As in the case of MEPG, the first two can be G(PO)MDP estimators (3.77) and a similar unbiased estimator can be derived for the meta gradient (Papini et al., 2020).

We make the following assumption on the gradient estimators,[5] which are reasonable for G(PO)MDP as shown in Section 5.5:

Assumption 8.1 *For every $\delta \in (0, 1)$ there exists a non-negative constant ϵ_δ such that, with probability at least $1 - \delta$:*

$$\left\| \nabla_{\boldsymbol{v}} J(\boldsymbol{v}, \omega) - \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}, \omega) \right\| \leqslant \frac{\epsilon_\delta}{\sqrt{N}},$$
$$\left\| \nabla_{\omega} J(\boldsymbol{v}, \omega) - \widehat{\nabla}_{\omega} J(\boldsymbol{v}, \omega) \right\| \leqslant \frac{\epsilon_\delta}{\sqrt{N}},$$

for every $\boldsymbol{v} \in \mathbb{R}^d$, $\omega \in \mathbb{R}$ and $N \geqslant 1$.

Here ϵ_δ represents an upper bound on the gradient estimation error, which can be characterized using various statistical inequalities (Section 5.8). Let us first compute the largest safe step size for mean updates:

Lemma 8.2 *Consider the class of Gaussian policies parametrized as in* (8.7)*. Let $\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \alpha_k \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega)$ with $\alpha_k \geqslant 0$ and $\omega_{k+1} = \omega_k$. Under Assumption 8.1, provided $N > \epsilon_\delta^2 / \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) \right\|^2$, for any $C_k \geqslant 0$, the largest step size guaranteeing Condition 8.1 with probability at least $1 - \delta$ is:*

$$\widetilde{\alpha}_k = \frac{\sigma_{\omega_k}^2 \Delta_k}{F \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k) \right\|} \left(1 + \sqrt{1 + \frac{2FC_k}{\sigma_{\omega_k}^2 \Delta_k^2}} \right),$$

[5] We do not need a similar assumption on the meta-gradient estimator $\widehat{\nabla}_{\omega} \mathcal{L}$, since our improvement requirements are always on the expected return.

where $\Delta_k = \left\|\widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega_k)\right\| - \frac{\epsilon_\delta}{\sqrt{N}}$ *and* F *is from* (8.8).

Proof Let denote ω_k simply as ω and $L = F/\sigma_\omega^2$ to reduce cluttering. From Theorem 5.2 and (5.9):

$$
\begin{aligned}
J(\boldsymbol{v}_{k+1}, \omega) - J(\boldsymbol{v}_k, \omega) &\geqslant \alpha_k \left\langle \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega), \nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\rangle - \alpha_k^2 \frac{L}{2} \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\|^2 \\
&= \alpha_k \left\langle \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega), \nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \pm \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\rangle \\
&\quad - \alpha_k^2 \frac{L}{2} \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\|^2 \\
&= \alpha_k \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\|^2 + \alpha_k \left\langle \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega), \nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) - \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\rangle \\
&\quad - \alpha_k^2 \frac{L}{2} \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\|^2 \\
&\geqslant \alpha_k \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\|^2 - \alpha_k \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\| \left\| \nabla_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) - \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\| \\
&\quad - \alpha_k^2 \frac{L}{2} \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\|^2 \qquad (8.22) \\
&\geqslant \alpha_k \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\|^2 - \alpha_k \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\| \frac{\epsilon_\delta}{\sqrt{N}} \\
&\quad - \alpha_k^2 \frac{L}{2} \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\|^2 \qquad (8.23) \\
&\geqslant \alpha_k \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\| \Delta_k - \alpha_k^2 \frac{L}{2} \left\| \widehat{\nabla}_{\boldsymbol{v}} J(\boldsymbol{v}_k, \omega) \right\|^2 := f(\alpha_k),
\end{aligned}
$$

where (8.22) is from Cauchy-Schwartz inequality and (8.23) is from Assumption 8.1. The hypothesis on the batch size ensures Δ_k is positive. To complete the proof, we solve $f(\alpha_k) \geqslant -C_k$ for α_k an retain the largest solution. ■

Similarly, we can compute the largest safe step size for *exploratory* variance updates:

Lemma 8.3 *Consider the class of Gaussian policies parametrized as in* (8.7)*. Let* $\omega_{k+1} = \omega_k + \eta_k \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k)$ *with* $\eta_k \geqslant 0$ *and* $\boldsymbol{v}_{k+1} = \boldsymbol{v}_k$*. Under Assumption 8.1, provided* $\widehat{\lambda}_k \neq 0$ *and* $N > \epsilon_\delta^2 / \widehat{\lambda}_k^2$*, for any* $C_k \geqslant 0$*, the largest step-size guaranteeing Condition 8.1 with probability at least* $1 - \delta$ *is:*

$$
\widetilde{\eta}_k = \frac{\left|\widetilde{\Delta}_k\right|}{G \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}_k, \omega_k) \right\|} \left(\operatorname{sign}\left(\widehat{\lambda}_k\right) + \sqrt{1 + \frac{2GC_k}{\widetilde{\Delta}_k^2}} \right),
$$

where $\widetilde{\Delta}_k = \widehat{\lambda}_k - \epsilon_\delta / \sqrt{N}$*,* $\widehat{\lambda}_k := \left\langle \widehat{\nabla}_\omega J(\boldsymbol{v}_k, \omega_k), \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}_k, \omega_k) \right\rangle \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}_k, \omega_k) \right\|^{-1}$ *is the scalar projection of* $\widehat{\nabla}_\omega J(\boldsymbol{v}_k, \omega_k)$ *onto* $\widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}_k, \omega_k)$*, and* G *is from* (8.14)*. If* $\widetilde{\lambda}_k = 0$*, select* $\widetilde{\eta}_k = \sqrt{\frac{2C_k}{G}} \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}_k, \omega_k) \right\|^{-1}$ *for any* $N \geqslant 1$ *instead.*

Proof Let $\boldsymbol{v}_k = \boldsymbol{v}$ to reduce clutter. First assume $\widehat{\lambda}_k \neq 0$. From Theorem 5.2 and Lemma 8.1:

$$\begin{aligned}
J(\boldsymbol{v}, \omega_{k+1}) - J(\boldsymbol{v}, \omega_k) &\geqslant \eta_k \left\langle \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k), \nabla_\omega J(\boldsymbol{v}, \omega_k) \right\rangle - \eta_k^2 \frac{G}{2} \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k) \right\|^2 \\
&= \eta_k \left\langle \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k), \nabla_\omega J(\boldsymbol{v}, \omega_k) \pm \widehat{\nabla}_\omega J(\boldsymbol{v}, \omega_k) \right\rangle - \eta_k^2 \frac{G}{2} \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k) \right\|^2 \\
&= \eta_k \left\langle \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k), \widehat{\nabla}_\omega J(\boldsymbol{v}, \omega_k) \right\rangle \\
&\quad + \eta_k \left\langle \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k), \nabla_\omega J(\boldsymbol{v}, \omega_k) - \widehat{\nabla}_\omega J(\boldsymbol{v}, \omega_k) \right\rangle \\
&\quad - \eta_k^2 \frac{G}{2} \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k) \right\|^2 \\
&\geqslant \eta_k \left\langle \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k), \widehat{\nabla}_\omega J(\boldsymbol{v}, \omega_k) \right\rangle \\
&\quad - \eta_k \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k) \right\| \left\| \nabla_\omega J(\boldsymbol{v}, \omega_k) - \widehat{\nabla}_\omega J(\boldsymbol{v}, \omega_k) \right\| \\
&\quad - \eta_k^2 \frac{G}{2} \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k) \right\|^2 \qquad (8.24) \\
&\geqslant \eta_k \left\langle \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k), \widehat{\nabla}_\omega J(\boldsymbol{v}, \omega_k) \right\rangle - \eta_k \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k) \right\| \frac{\epsilon_\delta}{\sqrt{N}} \\
&\quad - \eta_k^2 \frac{G}{2} \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k) \right\|^2 \qquad (8.25) \\
&\geqslant \eta_k \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k) \right\| \widetilde{\Delta}_k - \eta_k^2 \frac{G}{2} \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}, \omega_k) \right\|^2 := f(\eta_k),
\end{aligned}$$

where (8.24) is from Cauchy-Schwartz inequality and (8.25) is from Assumption 8.1. Solving $f(\eta_k) \geqslant -C_k$ for η_k and selecting the largest solution yields:

$$\widetilde{\eta}_k = \frac{\left| \widetilde{\Delta}_k \right|}{G \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}_k, \omega_k) \right\|} \left(\operatorname{sign}\left(\widetilde{\Delta}_k \right) + \sqrt{1 + \frac{2GC_k}{\widetilde{\Delta}_k^2}} \right). \qquad (8.26)$$

Under the batch-size condition, this is equivalent to:

$$\widetilde{\eta}_k = \frac{\left| \widetilde{\Delta}_k \right|}{G \left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}_k, \omega_k) \right\|} \left(\operatorname{sign}\left(\widehat{\lambda}_k \right) + \sqrt{1 + \frac{2GC_k}{\widetilde{\Delta}_k^2}} \right). \qquad (8.27)$$

Indeed, this is always non-negative.

As in Theorem 8.2, we can treat the case $\widehat{\lambda}_k = 0$ separately to obtain:

$$\widetilde{\eta}_k = \frac{1}{\left\| \widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{v}_k, \omega_k) \right\|} \sqrt{\frac{2C_k}{G}}, \qquad (8.28)$$

No assumption on the batch size is requested in this case. ■

Algorithm 11 SEPG (Safely Exploring Policy Gradient)

Input: Initial parameters $\boldsymbol{\upsilon}_0$ and ω_0, batch size N, sequence of slacks $\{C_k\}_{t=0}^{\infty}$, confidence parameter δ
for $k = 0, 1, \ldots$ **do**
  Collect a batch of N trajectories with $\pi_{\boldsymbol{\upsilon}_k, \omega_k}$
  Estimate $\widehat{\nabla}_{\boldsymbol{\upsilon}} J(\boldsymbol{\upsilon}_k, \omega_k)$,
  if t is even **then**
    if $N \leqslant \epsilon_\delta^2 / \left\| \widehat{\nabla}_{\boldsymbol{\upsilon}} J(\boldsymbol{\upsilon}_k, \omega_k) \right\|^2$ **then return**
    $\boldsymbol{\upsilon}_{k+1} = \boldsymbol{\upsilon}_k + \widetilde{\alpha}_k \nabla_{\boldsymbol{\upsilon}} J(\boldsymbol{\upsilon}_k, \omega_k)$ ▷ Safe step size from Lemma 8.2
  else
    estimate $\widehat{\nabla}_\omega J(\boldsymbol{\upsilon}_k, \omega_k)$ and $\widehat{\nabla}_\omega \left\| \widehat{\nabla}_{\boldsymbol{\upsilon}} J(\boldsymbol{\upsilon}_k, \omega_k) \right\|$
    if $N \leqslant \epsilon_\delta^2 / \widehat{\lambda}_k^2$ **then return** ▷ Projection $\widehat{\lambda}$ from Lemma 8.3
    $\widehat{\nabla}_\omega \mathcal{L}(\boldsymbol{\upsilon}_k, \omega_k) = \widehat{\nabla}_\omega J(\boldsymbol{\upsilon}_k, \omega_k) + \widetilde{\alpha}_k \Big(2 \left\| \widehat{\nabla}_{\boldsymbol{\upsilon}} J(\boldsymbol{\upsilon}_k, \omega_k) \right\| + \widehat{\nabla}_\omega \left\| \nabla_{\boldsymbol{\upsilon}} J(\boldsymbol{\upsilon}_k, \omega_k) \right\| \Big)$
    $\omega_{k+1} = \omega_k + \widetilde{\eta}_k \nabla_\omega \mathcal{L}(\boldsymbol{\upsilon}_k, \omega_k)$ ▷ Safe step size from Lemma 8.3
  end if
end for

Our *Safely Exploring Policy Gradient* (SEPG) algorithm is simply a variant of MEPG (Section 8.2.1) that employs these safe step sizes. Pseudo-code is provided in Algorithm 11.

As intended, the choice of step sizes guarantees bounded performance worsening (8.1) at each update:

Theorem 8.3 *Under Assumption 8.1, for any $k \geqslant 0$, provided $C_k \geqslant 0$, $N > \epsilon_\delta^2 / \left\| \widehat{\nabla}_{\boldsymbol{\upsilon}} J(\boldsymbol{\upsilon}_k, \omega_k) \right\|^2$ for even k, and $N > \epsilon_\delta^2 / \widehat{\lambda}_k^2$ for odd k, Algorithm 11 guarantees $J(\boldsymbol{\theta}_{k+1}) - J(\boldsymbol{\theta}_k) \geqslant C_k$ with probability at least $1 - \delta$.*

Proof We simply combine the results from Lemmas 8.2 and 8.3. Note how the alternating update schedule is necessary to apply them. ■

In the pseudo-code we have considered a simple version were the user selects a fixed batch size N and the algorithm stops as soon as N does not satisfy the requirements.

8.3.3 Empirical evaluation of SEPG

In this section we evaluate the SEPG algorithm on continuous control tasks.

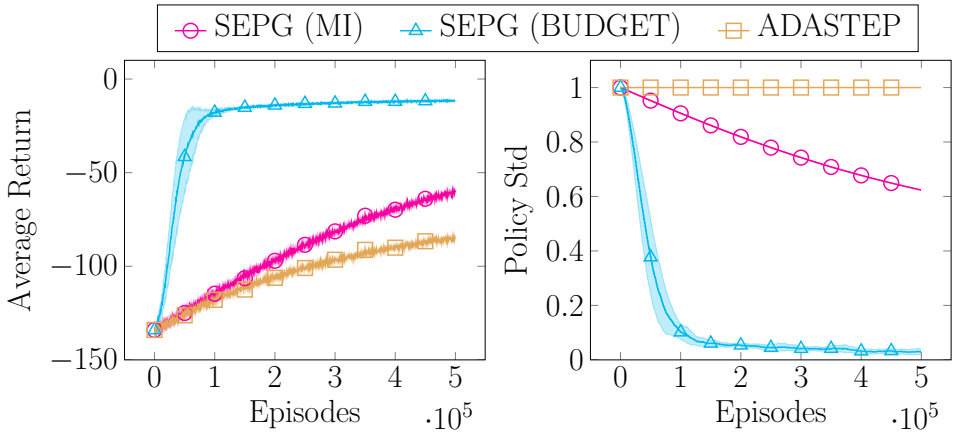


Figure 8.3: Average return (undiscounted) and policy standard deviation per episode of SEPG and ADASTEP on the LQR task, averaged over 10 independent runs with 95% Student's t-confidence intervals.

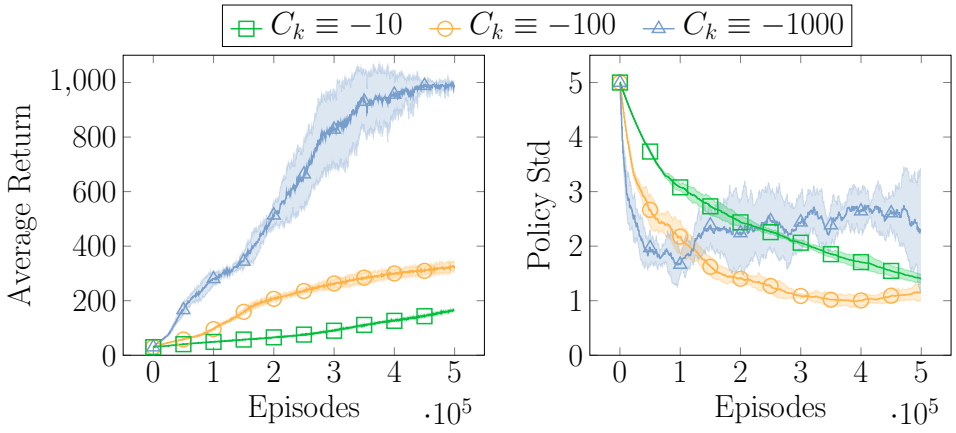


Figure 8.4: Average return (undiscounted) and standard deviation per episode of SEPG on the Cart-Pole task for different values of C_k, averaged over 5 runs with 95% confidence intervals.

Figure 8.3 shows the performance and the policy standard deviation of SEPG on the one-dimensional LQR task (Section C.1). SEPG with a monotonic improvement constraint ($C_k \equiv 0$) is compared with the adaptive-step-size algorithm (ADASTEP in the figure) by Pirotta et al. (2013a). Starting from $\sigma_{\omega_0} = 1$, SEPG achieves higher returns by safely lowering it, while ADASTEP has no way to safely update this parameter. Both algorithms use $\delta = 0.2$ and a large batch size ($N = 500$). We also consider a looser constraint, already discussed in Section 8.1.1 (BUDGET in the figure): that of never doing worse than the initial performance ($C_k = J(\boldsymbol{\theta}_0) - J(\boldsymbol{\theta}_k)$).

As expected, this allows faster learning, leading to optimal performance within a reasonable time.

On the Cart-Pole task, MEPG showed an oscillatory behavior. Motivated by this fact, we run SEPG with a fixed, positive slack C_k. Recall that the meaning of such a constraint is to limit per-update performance worsening. Figure 8.4 shows the results for different values of the threshold, starting from $\sigma_{\omega_0} = 5$ and neglecting the gradient estimation error (i.e., $\delta = 1$). Even under this simplifying assumption, only a very large value of C_k allows to reach optimal performance within a reasonable time. This is due to the worst-case assumptions of SEPG, which leads to over-conservative step sizes. Similarly to the penalty coefficient in POIS (Chapter 6), the threshold C_k, deprived of its precise meaning, could be tuned as a meta-parameter for specific applications. Note how oscillations are reduced w.r.t. MEPG, and how policy standard deviation is first reduced and then *safely increased* again.

We will discuss open questions regarding MEPG and SEPG in the coming conclusion.

Conclusion 9

We conclude this dissertation by summarizing our contributions, discussing them further and providing some ideas for future work.

Motivated by the potential application of policy optimization algorithms to real-world control problems, we have presented the PO framework from its theoretical foundations to the main algorithmic solutions (Chapters 2 and 3). At the same time, we have reviewed the main safety concerns arising from training physical agents, or even software agents that can have concrete, potentially catastrophic consequences to their environment (Chapter 4). It is by no coincidence that we focused on the safety of policy optimization algorithms, since the methods that are more likely to be applied to real-life problems are also the ones that more urgently need safety guarantees. Building reliable reinforcement learning agents for realistic settings is a complex task that will require both a deep theoretical understanding of learning algorithms and careful engineering. We approached the problem from the theoretical side, and focused on establishing formal guarantees for policy gradient algorithms.

Monotonic Improvement. Safety is a multi-faceted problem, and we have only touched a few aspects. The main perspective adopted in this manuscript is that of *monotonic performance improvement* (Chapter 5), which requires the agent to steadily improve its behavior during the learning process. This is a problem of *Seldonian machine learning* (Thomas et al., 2019): however risk is encoded in the optimization objective for a specific task, only careful algorithmic design can ensure that the agent does not learn unsafe behavior due to its partial knowledge of the environment. It is, at the same time, a problem of *safe exploration*, since every policy implemented by the agent can have concrete consequences. In on-policy learning, this includes all intermediate solutions.

To establish monotonic improvement bounds for policy gradient algorithms, we built upon previous work by Kakade (2001a), Pirotta et al. (2013a), and our previous work (Papini et al., 2017). We extended existing results in two main ways: we significantly expanded the set of policies for which MI can be enforced from Gaussian to *smoothing policies*, and we provided MI guarantees for stochastic gradient updates. The proposed algorithm, SPG, is an adaptive batch-size variant of REINFORCE, a more general version of the one proposed in (Papini et al., 2017). Differently from Schulman et al. (2015a), monotonic improvement can actually be guaranteed with high probability in practice, without approximations of sort, only knowing an upper bound on rewards and input variables. In turn, we loose some generality and all the practicality (we restore some of the latter with the SSPG variant). However, we believe this kind of result is an important link between theory and practice, showing the potentiality *and* limitations of policy gradient methods.

Future work should build upon these theoretical analysis to devise practical policy gradient algorithms with reasonably reliable improvement properties. Another important direction consists in combining this concept of safety with other ones, for instance by considering explicit constraints or risk-averse objectives (Bisi et al., 2020). Our results could also be applied to deep learning, including supervised and unsupervised problems. Smoothness of the objective function could be enforced through spectral normalization (Miyato et al., 2018). Notice that outside of real-world RL, where optimization on several data batches can be carried out in parallel, a more sophisticated trade-off for the batch size may be needed that takes into account specific computing architectures and parallelization techniques. From the theoretical side, an important open question remains about the sample complexity of SPG, given that it employs an increasing batch size. The techniques developed in Chapter 7 cannot be used as-is for two reasons: (1) they consider fixed batch sizes, and (2) they reason in expectation over the whole learning process, while SPG was explicitly designed to have per-update high-probability guarantees. For these reasons, further work is required.

Semi-offline learning (POIS). The fundamental role of the batch size for safe policy updates, and the great number of samples required even for toy problems, testify the paramount importance of the sample complexity problem.

In SPG, every intermediate policy interacts with the environment to collect fresh data for the next update. Using the same data for multiple policy updates can greatly improve the sample complexity. This is an off-policy learning problem, since with every update the target distribution goes further and further from the one used to collect the data. In Chapter 6, we studied the orthogonal problem of making a responsible use of data in off-policy optimization. We reviewed *importance sampling* techniques and showed how the distributional-shift problem mentioned above can lead to overconfident policy updates and unpredictable behavior. The key to develop a conservative off-policy learning strategy was a measure of the variance of importance weights suggested by Cortes et al. (2010), the Rényi divergence. By adding an explicit Rényi penalty to the expected return objective estimated with importance sampling, we can lower-bound the actual performance with high

probability. Our POIS algorithm optimizes this lower bound with policy gradients. We propose variants of POIS for parameter-based exploration and integrate advanced importance sampling techniques to improve sample complexity even further. By turning the penalization coefficient into a meta-parameter, we achieve state-of-the-art performance on benchmark control tasks. The downside of this practical approach is that the confidence parameter from the original statistical inequalities loses its meaning.

The two flavors of POIS have separate issues, both incarnations of the curse of dimensionality, which should be the main focus of future work. The action-based version, which is a generalization of REINFORCE, is subject to the *curse of horizon*: the variance of importance weights grows exponentially with the length of the task episodes. This could be avoided by weighting single transitions, but finding the correct importance weights is much more challenging in that case. Recent work (Hallak and Mannor, 2017; Liu et al., 2018b; Gelada and Bellemare, 2019; Liu et al., 2019c) tries to break this curse of horizon by estimating the ratio of induced state distributions. Applying this kind of approach to a partially-offline algorithm like POIS looks like a promising direction. Parameter-based POIS, as a generalization of PGPE, is too black-box to be affected by the length of the task horizon. However, it incurs in similar problems when the number of low-level policy parameters is very high, for instance with deep neural policies. In this case, compact representations of low-level policies, in the spirit of the fingerprinting techinque proposed by Harb et al. (2020) may prove fundamental.

Since we sacrifice the statistical meaning of the confidence parameter δ anyway, we could alternatively settle for guarantees in expectation, paired with a constraint on the variance of the policy updates. This would lead to an algorithm similar to TRPO (Schulman et al., 2015a), but based on the Rényi divergence in place of the less conservative KL divergence. As discussed in Section 6.2.4, a second-order approximation of the constraint leads to an adaptive natural policy gradient algorithm in both cases. The separation of target and behavioral policies operated by POIS opens another interesting research direction: that of explicitly learning behavioral policies with good exploration capabilities (Hanna and Stone, 2019). This also raises safe-exploration concerns like the one studied in Chapter 8. Finally, we wonder if the conservative approach of POIS could benefit other settings, such as actor-critic algorithms or black-box optimization.

Convergence. We studied the convergence properties of policy gradient algorithms for two main reasons. First, convergence is the unavoidable precondition of any safe algorithm. Second, it allows a rigorous characterization of the sample complexity of these algorithms. Arguably, no one would implement a learning algorithm on a physical agent without guarantees of convergence and without having an idea of the total amount of interaction it will require.

We first revisited some well known results on REINFORCE and PGT, by discussing the assumptions on which they rely. The concept of *smoothing policies* introduced in Chapter 5 proved useful to relax some of these requirements. Our main contribution, however, was a variant of REINFORCE based on SVRG, a

semi-stochastic gradient descent technique. Understanding the challenges of using semi-stochastic gradients in policy optimization proved fundamental to design a sound SVRPG algorithm and to identify a reasonable set of assumptions for proving convergence guarantees. We report a proof by (Xu et al., 2019) showing that SVRPG has indeed better sample complexity than REINFORCE. We also review later semi-stochastic policy gradient methods with even better sample complexity.

The rate proved for these latter algorithms is believed to be optimal. A first important future work is to establish formal lower bounds for the policy optimization problem, possibly starting from analogous results in stochastic nonconvex optimization (Arjevani et al., 2019). Another relevant direction consists in removing or relaxing some of the assumptions. Methods that rely on importance sampling, like SVRPG, are also subject to the "curse of horizon" problem discussed for action-based POIS, and would equally benefit from integrating work like (Liu et al., 2018b). Outright removing the need of importance weights is also a viable solution (Shen et al., 2019). Relaxing the requirements on the policy, for instance by using the concept of *smoothing policies* as done for REINFORCE, seems more challenging.

Safe exploration. Chapter 8.1.4 can be seen as an extension of the results of Chapter 5 to the problematic case of Gaussian policies with adaptive variance. We have situated it at the end of the dissertation for two reasons: first, the challenges of learning the policy variance emerged from all the preceding chapters. Second, we use this special case as a proxy for understanding the more general problem of reconciling safety and exploration, or of finding the right trade-off between epistemic and active risk, which is the natural last stop of our journey through safe policy optimization.

The problem of regulating exploration from the limited information available to a policy gradient algorithm is itself very challenging. Our MEPG heuristic employs a surrogate objective for the policy variance that tries to capture the long-term advantages of stochastic behavior. This requires to estimate some second-order information, which is not for free. Whether this technique can be reliably applied to larger-scale problems is a matter that should be further investigated. The SEPG algorithm is the adaptive step-size version of MEPG and comes with monotonic improvement guarantees of the same kind of those from Chapter 5. The step size for the policy variance is defined in a way that allows to follow the surrogate exploratory objective only when this does not affect immediate performance too much. This result offers some insights on the aforementioned trade-off between minimizing immediate risk due to the agent's own stochasticity and long-term epistemic risk that can benefit from the diverse data offered by random exploration. We could even say it fully captures the safe-exploration Seldonian problem that we framed in Chapter 4, albeit cast to a very specific setting. Similarly to Chapter 6, we also show how manually relaxing safety requirements via meta-parameters can lead to the desired practical results.

Future work should investigate whether this kind of adaptive exploration can be applied to more general settings, starting from other policy classes. The convergence properties of MEPG represent another important open problem. Finally, random

exploration may not be acceptable in some real-world settings for practical or ethical reasons. An other important research direction consists in developing effective forms of exploration that do not rely on action randomization. This comes with the challenge of extending existing safety guarantees to deterministic policies, which do not possess the smoothing property that was so central to our theory. We believe that this can only be done by making some assumptions on the regularity of the environment, like in (Pirotta et al., 2015).

Many open questions remain. For instance, we have completely neglected model-based policy optimization (Nguyen-Tuong and Peters, 2011; Deisenroth et al., 2013), which could play an important role in improving the safety of RL agents. With a model of the environment, the agent can more easily predict the effects of its actions and avoid hazards. An example of recent work on the topic is (Berkenkamp et al., 2017). We have also focused on agents learning from scratch in the real world, but many practical problems would probably benefit from a mix of simulation and real experience (e.g., Büchler et al., 2020; Peng et al., 2020a). From the theoretical perspective, global optimality guarantees for policy gradient algorithms are now an active area of research (Section 7.7) which could have a big impact on policy optimization algorithms in the near future.

To conclude, we have highlighted the potential and the limitations of policy optimization algorithms in view of applying them to real-world scenarios where the behavior of the agent during the learning phase is necessarily subject to safety constraints and collecting data is an expensive and risky process. We have developed the theory of policy gradients establishing novel theoretical guarantees of monotonic performance improvement and convergence. We have used some of these theoretical findings to design policy gradient algorithms with more desirable properties than previously available ones, and tested them on benchmark tasks to support theory with empirical evaluations. We have followed as much as possible the Seldonian principle of designing rigorously safe algorithms, letting the user decide the amount of compromise between safety guarantees and practical efficiency. POIS is a good example of how a theoretically sound algorithm can obtain competitive performances by relaxing the safety requirements.

In general, the gap between theory and practice is still large, and further work is required before policy optimization can be safely applied to real-life problems. We hope that our effort will guide the responsible implementation of existing algorithms, and inspire the development of even better ones.

Smooth Functions A

In this Appendix, we review the properties of smooth functions.

Let $g : \mathcal{X} \subseteq \mathbb{R}^m \to \mathbb{R}^n$ be a (possibly non-convex) vector-valued function. We call g *Lipschitz continuous* if there exists $L > 0$ such that, for every $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$:

$$\left\|g(\boldsymbol{x}') - g(\boldsymbol{x})\right\| \leqslant L \left\|\boldsymbol{x}' - \boldsymbol{x}\right\| . \tag{A.1}$$

Let $f : \mathcal{X} \subseteq \mathbb{R}^m \to \mathbb{R}$ be a real-valued differentiable function. We call f *Lipschitz smooth* if its gradient is Lipschitz continuous, i.e., there exists $L > 0$ such that, for every $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$:

$$\left\|\nabla f(\boldsymbol{x}') - \nabla f(\boldsymbol{x})\right\| \leqslant L \left\|\boldsymbol{x}' - \boldsymbol{x}\right\| . \tag{A.2}$$

Whenever we want to specify the Lipschitz constant[1] L of the gradient, we call f L-smooth. For a twice-differentiable function, the following holds[2]:

Lemma A.1 *Let $\mathcal{X}$ be a convex subset of $\mathbb{R}^m$ and $f : \mathcal{X} \to \mathbb{R}$ be a twice-differentiable function. If the Hessian is uniformly bounded in spectral norm by $L > 0$, i.e., $\sup_{\boldsymbol{x} \in \mathcal{X}} \left\|\nabla^2 f(\boldsymbol{x})\right\|_2 \leqslant L$, f is L-smooth.*

Proof Let $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$, $\boldsymbol{h} \coloneqq \boldsymbol{x}' - \boldsymbol{x}$ and $g : [0, 1] \to \mathbb{R}$, $g(\lambda) \equiv \nabla_{\boldsymbol{x}} f(\boldsymbol{x} + \lambda \boldsymbol{h})$. Convexity of $\mathcal{X}$ guarantees $\boldsymbol{x} + \lambda \boldsymbol{h} \in \mathcal{X}$ for $\lambda \in [0, 1]$. Twice-differentiability of f implies $\nabla_{\boldsymbol{x}} f$ is continuous, which in turn implies g is continuous. From the

[1] The Lipschitz constant is usually defined as the *smallest* constant satisfying the Lipschitz condition. In this paper, instead, we call Lipschitz constant *any* constant for which the Lipschitz condition holds.

[2] The results from this section are well known in the optimization literature. We report proofs for the sake of completeness.

Fundamental Theorem of Calculus:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}') - \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}+\boldsymbol{h}) - \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = g(1) - g(0) = \int_0^1 g'(\lambda)\,\mathrm{d}\lambda$$
$$= \int_0^1 \boldsymbol{h}^T \nabla^2_{\boldsymbol{x}} f(\boldsymbol{x}+\lambda\boldsymbol{h})\,\mathrm{d}\lambda. \tag{A.3}$$

Hence:

$$\begin{aligned}
\left\|\nabla_{\boldsymbol{x}} f(\boldsymbol{x}') - \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\right\| &= \left\| \int_0^1 \boldsymbol{h}^T \nabla^2_{\boldsymbol{x}} f(\boldsymbol{x}+\lambda\boldsymbol{h})\,\mathrm{d}\lambda \right\|_2 \\
&\leqslant \int_0^1 \left\| \nabla^2_{\boldsymbol{x}} f(\boldsymbol{x}+\lambda\boldsymbol{h})\boldsymbol{h} \right\|_2 \mathrm{d}\lambda \\
&\leqslant \int_0^1 \left\| \nabla^2_{\boldsymbol{x}} f(\boldsymbol{x}+\lambda\boldsymbol{h}) \right\|_2 \|\boldsymbol{h}\|_2 \,\mathrm{d}\lambda \qquad (A.4) \\
&\leqslant L\,\|\boldsymbol{h}\|_2 = L\left\|\boldsymbol{x}' - \boldsymbol{x}\right\|_2, \qquad (A.5)
\end{aligned}$$

where (A.4) is from the consistency of induced norms, i.e., $\|A\boldsymbol{x}\|_p \leqslant \|A\|_p \|\boldsymbol{x}\|_p$. ■

Lipschitz smooth functions admit a quadratic bound on the deviation from a linear behavior:

Lemma A.2 (Quadratic Bound) *Let $\mathcal{X}$ be a convex subset of $\mathbb{R}^m$ and $f : \mathcal{X} \to \mathbb{R}$ be an L-smooth function. Then, for every $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$:*

$$\left| f(\boldsymbol{x}') - f(\boldsymbol{x}) - \left\langle \boldsymbol{x}' - \boldsymbol{x}, \nabla f(\boldsymbol{x}) \right\rangle \right| \leqslant \frac{L}{2} \left\|\boldsymbol{x}' - \boldsymbol{x}\right\|^2, \tag{A.6}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product.

Proof Let $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$, $\boldsymbol{h} \coloneqq \boldsymbol{x}' - \boldsymbol{x}$ and $g : [0,1] \to \mathbb{R}$, $g(\lambda) \equiv f(\boldsymbol{x} + \lambda\boldsymbol{h})$. Convexity of $\mathcal{X}$ guarantees $\boldsymbol{x} + \lambda\boldsymbol{h} \in \mathcal{X}$ for $\lambda \in [0,1]$. Lipschitz smoothness implies continuity of f, which in turn implies g is continuous. From the Fundamental Theorem of Calculus:

$$f(\boldsymbol{x}') - f(\boldsymbol{x}) = g(1) - g(0) = \int_0^1 g'(\lambda)\,\mathrm{d}\lambda. \tag{A.7}$$

Hence:

$$\begin{aligned}
\left|f(\boldsymbol{x}') - f(\boldsymbol{x}) - \left\langle \boldsymbol{x}' - \ \boldsymbol{x}, \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\right\rangle\right| &= \left|\int_0^1 g'(\lambda)\,\mathrm{d}\lambda - \langle \boldsymbol{h}, \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\rangle\right| \\
&= \left|\int_0^1 \langle \boldsymbol{h}, \nabla_{\boldsymbol{x}} f(\boldsymbol{x}+\lambda\boldsymbol{h})\rangle\,\mathrm{d}\lambda - \langle \boldsymbol{h}, \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\rangle\right| \\
&= \left|\int_0^1 \langle \boldsymbol{h}, \nabla_{\boldsymbol{x}} f(\boldsymbol{x}+\lambda\boldsymbol{h}) - \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\rangle\,\mathrm{d}\lambda\right| \\
&\leqslant \int_0^1 |\langle \boldsymbol{h}, \nabla_{\boldsymbol{x}} f(\boldsymbol{x}+\lambda\boldsymbol{h}) - \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\rangle|\,\mathrm{d}\lambda \\
&\leqslant \int_0^1 \|\nabla_{\boldsymbol{x}} f(\boldsymbol{x}+\lambda\boldsymbol{h}) - \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\|_2 \,\|\boldsymbol{h}\|_2\,\mathrm{d}\lambda \qquad \text{(A.8)} \\
&\leqslant L\,\|\boldsymbol{h}\|_2^2 \int_0^1 \lambda\,\mathrm{d}\lambda \qquad \text{(A.9)} \\
&= \frac{L}{2}\,\|\boldsymbol{x}' - \boldsymbol{x}\|_2^2,
\end{aligned}$$

where (A.8) is from the Cauchy-Schwartz inequality and (A.9) is from the Lipschitz smoothness of f. ■

This bound is often useful for optimization purposes (Nesterov, 2004).

Auxiliary Lemmas B

In this Appendix, we provide some auxiliary Lemmas that were used in the proofs.

Lemma B.1 *Given two real sequences* $(x_i)_{i=0}^{\infty}$ *and* $(y_j)_{j=0}^{\infty}$*, for all* $n \geqslant 0$*:*

$$\sum_{i=0}^{n} x_i \sum_{j=0}^{i} y_j = \sum_{j=0}^{n} \left(\sum_{i=j}^{n} x_i \right) y_j. \tag{B.1}$$

Moreover,

$$\sum_{i=0}^{\infty} x_i \sum_{j=0}^{i} y_j = \sum_{j=0}^{\infty} \left(\sum_{i=j}^{\infty} x_i \right) y_j, \tag{B.2}$$

if the series converges.

Proof We prove the first statement by induction. For $n = 0$ we just have

$x_0 y_0 = x_0 y_0$. Now assume the statement holds for n:

$$\begin{aligned}
\sum_{i=0}^{n+1} x_i \sum_{j=0}^{i} y_j &= \sum_{i=0}^{n} x_i \sum_{j=0}^{i} y_j + x_{n+1} \sum_{j=0}^{n+1} y_j \\
&= \sum_{j=0}^{n} \left(\sum_{i=j}^{n} x_i \right) y_j + x_{n+1} \sum_{j=0}^{n+1} y_j && \text{(B.3)} \\
&= \sum_{j=0}^{n} \left(\sum_{i=j}^{n} x_i \right) y_j + x_{n+1} \sum_{j=0}^{n} y_j + x_{n+1} y_{n+1} \\
&= \sum_{j=0}^{n} \left(\sum_{i=j}^{n+1} x_i \right) y_j + x_{n+1} y_{n+1} \\
&= \sum_{j=0}^{n+1} \left(\sum_{i=j}^{n+1} x_i \right) y_j, && \text{(B.4)}
\end{aligned}$$

where 8.22 is from the inductive hypothesis. The second statement is obtained by taking the limit for $n \to \infty$ on both sides of (B.1). ■

Lemma B.2 *For all $\pi \in \Delta_{\mathcal{A}}^{\mathcal{S}}$, $s_0 \in \mathcal{S}$ and measurable $\mathcal{E} \subseteq \mathcal{S}$:*

$$d_{s_0}^{\pi}(\mathcal{E}) = (1-\gamma) p_{\pi}(\mathcal{E}|s) + \gamma \int_{\mathcal{S}} d_{s_0}^{\pi}(s) p(\mathcal{E}|s) \, \mathrm{d}s.$$

Proof

$$\begin{aligned}
\gamma \int_{\mathcal{S}} d_{s_0}^{\pi}(s) p(\mathcal{E}|s) \, \mathrm{d}s &= (1-\gamma) \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^t p_{\pi}^t(s|s_0) p_{\pi}(\mathcal{E}|s) \, \mathrm{d}s && \text{(B.5)} \\
&= (1-\gamma) \sum_{t=1}^{\infty} \gamma^t \int_{\mathcal{S}} p_{\pi}^t(s|s_0) p_{\pi}(\mathcal{E}|s) \, \mathrm{d}s && \text{(B.6)} \\
&= \frac{(1-\gamma)}{\gamma} \sum_{t=1}^{\infty} \gamma^{t+1} \int_{\mathcal{S}} p_{\pi}^t(s|s_0) p_{\pi}(\mathcal{E}|s) \, \mathrm{d}s && \text{(B.7)} \\
&= \frac{(1-\gamma)}{\gamma} \sum_{t=1}^{\infty} \gamma^{t+1} p_{\pi}^{t+1}(\mathcal{E}|s_0) && \text{(B.8)} \\
&= \frac{(1-\gamma)}{\gamma} \sum_{t=2}^{\infty} \gamma^{t} p_{\pi}^{t}(\mathcal{E}|s_0) && \text{(B.9)} \\
&= \frac{(1-\gamma)}{\gamma} \sum_{t=1}^{\infty} \gamma^{t} p_{\pi}^{t}(\mathcal{E}|s_0) - (1-\gamma) p_{\pi}(\mathcal{E}|s_0) && \text{(B.10)} \\
&= d_{s_0}^{\pi}(\mathcal{E}) - (1-\gamma) p_{\pi}(\mathcal{E}|s_0). && \text{(B.11)}
\end{aligned}$$

■

In this Appendix, we provide detailed descriptions of the tasks used in the numerical simulations. The horizon and the discount factor depend on the specific experiment and are reported in the pertaining sections. The same holds for LQR specifics.

C.1 LQR

The *Linear Quadratic Regulator* (LQR) is a classical optimal control problem (Dorato et al., 1994). It models the very general task of controlling a set of variables to zero with the minimum effort. Given a state space $\mathcal{S} \subseteq \mathbb{R}^n$ and an action space $\mathcal{A} \subseteq \mathbb{R}^m$, the next state is a linear function of current state and action:[1]

$$s_{t+1} = As_t + Ba_t, \tag{C.1}$$

where $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$. The reward is quadratic in both state and action:

$$r_{t+1} = s_t^T C s_t + a_t^T D a_t, \tag{C.2}$$

where $C \in \mathbb{R}^{n \times n}$ and $D \in \mathbb{R}^{m \times m}$ are positive definite matrices. A linear controller is optimal for this task (Dorato et al., 1994) and can be computed in closed form with dynamic-programming techniques. In our experiments, we always consider shallow Gaussian policies of the form:

$$\pi(\cdot|s_t) = \mathcal{N}(\boldsymbol{\theta}^T s_t, \sigma^2 \mathbb{I}), \tag{C.3}$$

[1] A zero-mean Gaussian noise is typically added to the next state to model disturbances. However, since we always consider Gaussian policies, we can ignore the system noise without loss of generality. Indeed, from linearity of the next state, said $\overline{a}_t$ the expected action under (C.3), $s_{t+1} = As_t + B\overline{a}_t + B\epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2 \mathbb{I})$. From the property of Gaussians, we can write $\epsilon = \epsilon_a + \epsilon_b$ where $\epsilon_a \sim \mathcal{N}(0, \sigma_a^2 \mathbb{I})$ is from the actual stochasticity of the agent and $\epsilon_b \sim \mathcal{N}(0, \sigma_b^2 B^\dagger)$ is the system noise, which can be subsumed by the policy noise in numerical simulations for simplicity.

where $\boldsymbol{\theta} \in \mathbb{R}^n$ and $\sigma > 0$ can be fixed or learned as an additional policy parameter. This version of LQR with Gaussian policies is also called LQG (Linear-Quadratic Gaussian Regulator, Peters and Schaal, 2008b). States and actions are clipped in practice when they happen to fall outside $\mathcal{S}$ and $\mathcal{A}$, respectively. We have ignored nonlinearities stemming from this fact.

C.2 Continuous-Action Cart-Pole Balancing

Cart-Pole balancing is a classical RL benchmark (Barto et al., 1983). We consider here a continuous-action variant.

This is a 2D continuous control task. The goal is to balance (keep upright) a pole situated on a cart, by applying forces to the cart in the horizontal direction. The cart has a mass of $1Kg$ and the pole has a mass of $0.1Kg$ and is $0.5m$ long. The state is 4-dimensional and includes the cart's position x, the cart's horizontal speed $\dot{x}$, the pole's angle w.r.t. the upright position θ and the pole's angular velocity $\dot{\theta}$. The action (force) that can be applied is $a \in [-10, 10]$. The agent receives a reward of 1 for each step. All state variables are initialized uniformly at random in $[-0.05, 0.05]$. The task is deterministic otherwise. The episode terminates when the pole falls ($|\theta| > 12$ degrees), when the cart goes too far from the initial position ($|x| > 2.4$), and anyway at the specified time horizon H. The control frequency is $50Hz$.

We have described here our own python implementation of the task,[2] employed in Chapter 8. In other cases (Chapters 6 and 7), for easier comparison with previous work, we consider the `rllab` version instead (see Section C.3 below). The main difference is that the agent receives a reward of 10 at each step in the `rllab` version. Hence, the optimal(undiscounted) return is $10H$ instead of H in this case.

C.3 Continuous Control Tasks from `rllab`

In Chapters 6 and 7 we evaluate our algorithms on continuous control tasks from the `rllab` library (`https://github.com/rll/rllab`). Refer to Duan et al. (2016) for details. We report here very brief descriptions. Here and in the rest of the manuscript, we use the task names from the implementation. We report in parentheses the names used by Duan et al. (2016) in their paper when different.

Cartpole (Cart-Pole Balancing): cart-pole balancing task like the one described in Section C.2. Four-dimensional states and one-dimensional actions.

Inverted Double Pendulum (Double Inverted Pendulum Balancing): like Cart-pole, but with a two-link pole. Five dimensional states and one-dimensional actions.

[2] `https://github.com/T3p/potion`.

Acrobot (Acrobot Swing Up): two-link robot arm. The first joint is fixed and only the second one is actuated. The goal is to swing-up the arm starting from a hanging position. Swing-up tasks require nonlinear controllers. Four-dimensional states and one-dimensional actions.

Mountain Car: an under-actuated car must escape a valley. Swinging up and down is necessary to build momentum. This is a classical exploration benchmark due to sparsity of the reward. Two-dimensional states and one-dimensional actions.

Inverted Pendulum (Cart-Pole Swingup): like Cartpole, but the pole is initially hanging and the agent needs to swing it up by moving the cart. Four-dimensional states and one-dimensional actions.

Swimmer: three-link snake robot in a fluid, must swim forward. Thirteen-dimensional states and two-dimensional actions.

Bibliography

Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *Int. J. Robotics Res.*, 29(13):1608–1639, 2010.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 22–31. PMLR, 2017.

Alekh Agarwal, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. Optimality and approximation with policy gradient methods in markov decision processes. In *COLT*, volume 125 of *Proceedings of Machine Learning Research*, pages 64–66. PMLR, 2020.

Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. Learning to generalize from sparse and underspecified rewards. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 130–140. PMLR, 2019.

Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 151–160. PMLR, 2019.

Zeyuan Allen-Zhu. Natasha 2: Faster non-convex optimization than SGD. In *NeurIPS*, pages 2680–2691, 2018.

Zeyuan Allen-Zhu and Elad Hazan. Variance reduction for faster non-convex optimization. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 699–707. JMLR.org, 2016.

Eitan Altman. Constrained markov decision processes with total cost criteria: Lagrangian approach and dual linear program. *Math. Methods Oper. Res.*, 48(3): 387–417, 1998.

Eitan Altman. *Constrained Markov decision processes*, volume 7. CRC Press, 1999.

Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

Shun-ichi Amari. *Information geometry and its applications*, volume 194. Springer, 2016.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul F. Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016.

Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *Int. J. Robotics Res.*, 39(1), 2020.

Yossi Arjevani, Yair Carmon, John C. Duchi, Dylan J. Foster, Nathan Srebro, and Blake E. Woodworth. Lower bounds for non-convex stochastic optimization. *CoRR*, abs/1912.02365, 2019.

Reza Babanezhad, Mohamed Osama Ahmed, Alim Virani, Mark Schmidt, Jakub Konecný, and Scott Sallinen. Stop wasting my gradients: Practical SVRG. *CoRR*, abs/1511.01942, 2015.

J. Andrew Bagnell and Jeff G. Schneider. Covariant policy search. In *IJCAI*, pages 1019–1024. Morgan Kaufmann, 2003.

Leemon C. Baird III. Residual algorithms: Reinforcement learning with function approximation. In *ICML*, pages 30–37. Morgan Kaufmann, 1995.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.*, 13(5):834–846, 1983.

Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *J. Artif. Intell. Res.*, 15:319–350, 2001.

Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Oper. Res. Lett.*, 31(3):167–175, 2003.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

Bernard Bercu, Bernard Delyon, and Emmanuel Rio. Concentration inequalities for sums. In *Concentration Inequalities for Sums and Martingales*, pages 11–60. Springer, 2015.

Felix Berkenkamp. *Safe Exploration in Reinforcement Learning: Theory and Applications in Robotics*. PhD thesis, ETH Zurich, 2019.

Felix Berkenkamp, Angela P. Schoellig, and Andreas Krause. Safe controller optimization for quadrotors with gaussian processes. In *ICRA*, pages 491–496. IEEE, 2016.

Felix Berkenkamp, Matteo Turchetta, Angela P. Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *NeurIPS*, pages 908–918, 2017.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

Dimitir P Bertsekas and Steven Shreve. *Stochastic optimal control: the discrete-time case*. 2004.

Dimitri P Bertsekas. Approximate policy iteration: A survey and some new methods. *Journal of Control Theory and Applications*, 9(3):310–335, 2011.

Dimitri P Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific Belmont, MA, 2019.

Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

Jalaj Bhandari and Daniel Russo. Global optimality guarantees for policy gradient methods. *CoRR*, abs/1906.01786, 2019.

Shalabh Bhatnagar and K. Lakshmanan. An online actor-critic algorithm with function approximation for constrained markov decision processes. *J. Optim. Theory Appl.*, 153(3):688–708, 2012.

Sarah Bird, Solon Barocas, Kate Crawford, Fernando Diaz, and Hanna Wallach. Exploring or exploiting? social and ethical implications of autonomous experimentation in ai. In *Workshop on Fairness, Accountability, and Transparency in Machine Learning*, 2016.

Lorenzo Bisi, Luca Sabbioni, Edoardo Vittori, Matteo Papini, and Marcello Restelli. Risk-averse trust region optimization for reward-volatility reduction. In *IJCAI*, pages 4583–4589. ijcai.org, 2020.

Vivek S. Borkar. An actor-critic algorithm for constrained markov decision processes. *Syst. Control. Lett.*, 54(3):207–213, 2005.

Vivek S Borkar and Vijaymohan R Konda. The actor-critic algorithm as multi-time-scale stochastic approximation. *Sadhana*, 22(4):525–543, 1997.

Nick Bostrom. *Superintelligence*. Dunod, 2017.

Léon Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

Léon Bottou and Yann LeCun. Large scale online learning. In *NeurIPS*, pages 217–224. MIT Press, 2003.

Justin A. Boyan. Technical update: Least-squares temporal difference learning. *Mach. Learn.*, 49(2-3):233–246, 2002.

Steven J. Bradtke. Reinforcement learning applied to linear quadratic regulation. In *NeurIPS*, pages 295–302. Morgan Kaufmann, 1992.

Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. *Mach. Learn.*, 22(1-3):33–57, 1996.

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

Sébastien Bubeck, Nicolò Cesa-Bianchi, and Gábor Lugosi. Bandits with heavy tail. *IEEE Trans. Inf. Theory*, 59(11):7711–7717, 2013.

Dieter Büchler, Simon Guist, Roberto Calandra, Vincent Berenz, Bernhard Schölkopf, and Jan Peters. Learning to play table tennis from scratch using muscular robots. *CoRR*, abs/2006.05935, 2020.

Jacob Burbea. The convexity with respect to gaussian distributions of divergences of order α. *Utilitas Mathematica*, 26:171–192, 1984.

Apostolos N. Burnetas and Michael N. Katehakis. Optimal adaptive policies for markov decision processes. *Math. Oper. Res.*, 22(1):222–255, 1997.

Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Syst. Man Cybern. Part C*, 38 (2):156–172, 2008.

Qi Cai, Zhuoran Yang, Chi Jin, and Zhaoran Wang. Provably efficient exploration in policy optimization. *CoRR*, abs/1912.05830, 2019a.

Qi Cai, Zhuoran Yang, Jason D. Lee, and Zhaoran Wang. Neural temporal-difference learning converges to global optima. In *NeurIPS*, pages 11312–11322, 2019b.

FP Cantelli. Sui confini della probabilita. In *Atti del Congresso Internazionale dei Matematici: Bologna del 3 al 10 de settembre di 1928*, pages 47–60, 1929.

A Castelletti, Stefano Galelli, Marcello Restelli, and Rodolfo Soncini-Sessa. Tree-based reinforcement learning for optimal water reservoir operation. *Water Resources Research*, 46(9), 2010.

Dotan Di Castro, Aviv Tamar, and Shie Mannor. Policy gradients with variance related risk criteria. In *ICML*. icml.cc / Omnipress, 2012.

Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

Mahmoud El Chamie, Yue Yu, and Behçet Açikmese. Convex synthesis of randomized policies for controlled markov chains with density safety upper bound constraints. In *ACC*, pages 6290–6295. IEEE, 2016.

Samuel P. M. Choi, Dit-Yan Yeung, and Nevin Lianwen Zhang. An environment model for nonstationary reinforcement learning. In *NeurIPS*, pages 987–993. The MIT Press, 1999.

Po-Wei Chou, Daniel Maturana, and Sebastian A. Scherer. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 834–843. PMLR, 2017.

Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *J. Mach. Learn. Res.*, 18:167:1–167:51, 2017.

Yinlam Chow, Ofir Nachum, Edgar A. Duéñez-Guzmán, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. In *NeurIPS*, pages 8103–8112, 2018.

Yinlam Chow, Ofir Nachum, Aleksandra Faust, Mohammad Ghavamzadeh, and Edgar A. Duéñez-Guzmán. Lyapunov-based safe policy optimization for continuous control. *CoRR*, abs/1901.10031, 2019.

Casey Chu, Jose H. Blanchet, and Peter W. Glynn. Probability functional descent: A unifying perspective on gans, variational inference, and reinforcement learning. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 1213–1222. PMLR, 2019.

Wesley Chung, Valentin Thomas, Marlos C. Machado, and Nicolas Le Roux. Beyond variance reduction: Understanding the true impact of baselines on policy optimization. *CoRR*, abs/2008.13773, 2020.

Kamil Ciosek and Shimon Whiteson. Expected policy gradients for reinforcement learning. *J. Mach. Learn. Res.*, 21:52:1–52:51, 2020.

Jeffery A Clouse. On integrating apprentice learning and reinforcement learning. 1997.

Jeffery A. Clouse and Paul E. Utgoff. A teaching method for reinforcement learning. In *ML*, pages 92–110. Morgan Kaufmann, 1992.

William G Cochran. *Sampling techniques*. John Wiley & Sons, 2007.

Andrew Cohen, Lei Yu, and Robert Wright. Diverse exploration for fast and safe policy improvement. In *AAAI*, pages 2876–2883. AAAI Press, 2018.

Andrew Cohen, Xingye Qiao, Lei Yu, Elliot Way, and Xiangrong Tong. Diverse exploration via conjugate policies for policy gradient methods. *CoRR*, abs/1902.03633, 2019.

Corinna Cortes, Yishay Mansour, and Mehryar Mohri. Learning bounds for importance weighting. In *NeurIPS*, pages 442–450. Curran Associates, Inc., 2010.

Ashok Cutkosky and Francesco Orabona. Momentum-based variance reduction in non-convex SGD. In *NeurIPS*, pages 15210–15219, 2019.

Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerík, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *CoRR*, abs/1801.08757, 2018.

Christian Daniel, Gerhard Neumann, Oliver Kroemer, and Jan Peters. Hierarchical relative entropy policy search. *J. Mach. Learn. Res.*, 17:93:1–93:50, 2016.

Aaron Defazio, Francis R. Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NeurIPS*, pages 1646–1654, 2014a.

Aaron Defazio, Justin Domke, and Tibério S. Caetano. Finito: A faster, permutable incremental gradient method for big data problems. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1125–1133. JMLR.org, 2014b.

Thomas Degris, Martha White, and Richard S. Sutton. Linear off-policy actor-critic. In *ICML*. icml.cc / Omnipress, 2012.

Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

Peter Dorato, Vito Cerone, and Chaouki Abdallah. *Linear-quadratic control: an introduction*. Simon & Schuster, Inc., 1994.

Simon S. Du, Jianshu Chen, Lihong Li, Lin Xiao, and Dengyong Zhou. Stochastic variance reduction methods for policy evaluation. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 1049–1058. PMLR, 2017.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1329–1338. JMLR.org, 2016.

Miroslav Dudík, John Langford, and Lihong Li. Doubly robust policy evaluation and learning. In *ICML*, pages 1097–1104. Omnipress, 2011.

Yonathan Efroni, Shie Mannor, and Matteo Pirotta. Exploration-exploitation in constrained mdps. *CoRR*, abs/2003.02189, 2020a.

Yonathan Efroni, Lior Shani, Aviv Rosenberg, and Shie Mannor. Optimistic policy optimization with bandit feedback. *CoRR*, abs/2002.08243, 2020b.

Benjamin Ellenberger. Pybullet gymperium. `https://github.com/benelot/pybullet-gym`, 2018–2019.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res.*, 6:503–556, 2005.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1406–1415. PMLR, 2018.

Alireza Fallah, Aryan Mokhtari, and Asuman E. Ozdaglar. Provably convergent policy gradient methods for model-agnostic meta-reinforcement learning. *CoRR*, abs/2002.05135, 2020.

Cong Fang, Chris Junchi Li, Zhouchen Lin, and Tong Zhang. SPIDER: near-optimal non-convex optimization via stochastic path-integrated differential estimator. In *NeurIPS*, pages 687–697, 2018.

Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. More robust doubly robust off-policy evaluation. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1446–1455. PMLR, 2018.

Maryam Fazel, Rong Ge, Sham M. Kakade, and Mehran Mesbahi. Global convergence of policy gradient methods for the linear quadratic regulator. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1466–1475. PMLR, 2018.

Matthew Fellows, Kamil Ciosek, and Shimon Whiteson. Fourier policy gradients. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1485–1494. PMLR, 2018.

K Ferentios. On tcebycheff's type inequalities. *Trabajos de Estadistica y de Investigacion Operativa*, 33(1):125, 1982.

Ronan Fruit, Alessandro Lazaric, and Matteo Pirotta. Regret minimization in infinite-horizon finite markov decision processes. Tutorial at ALT'19, 2019. URL `https://rlgammazero.github.io/`.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1582–1591. PMLR, 2018.

Yasuhiro Fujita and Shin-ichi Maeda. Clipped action policy gradient. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1592–1601. PMLR, 2018.

Thomas Furmston and David Barber. A unifying perspective of parametric policy search methods for markov decision processes. In *NeurIPS*, pages 2726–2734, 2012.

Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári. Multi-criteria reinforcement learning. In *ICML*, pages 197–205. Morgan Kaufmann, 1998.

Evrard Garcelon, Mohammad Ghavamzadeh, Alessandro Lazaric, and Matteo Pirotta. Conservative exploration in reinforcement learning. In *AISTATS*, volume 108 of *Proceedings of Machine Learning Research*, pages 1431–1441. PMLR, 2020.

Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.*, 16:1437–1480, 2015.

Matthieu Geist, Bruno Scherrer, and Olivier Pietquin. A theory of regularized markov decision processes. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2160–2169. PMLR, 2019.

Carles Gelada and Marc G. Bellemare. Off-policy deep reinforcement learning by bootstrapping the covariate shift. In *AAAI*, pages 3647–3655. AAAI Press, 2019.

Mohammad Ghavamzadeh, Marek Petrik, and Yinlam Chow. Safe policy improvement by minimizing robust baseline regret. In *NeurIPS*, pages 2298–2306, 2016.

M. Gil, Fady Alajaji, and Tamás Linder. Rényi divergence measures for commonly used univariate continuous distributions. *Inf. Sci.*, 249:124–131, 2013.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Geoffrey J. Gordon. Reinforcement learning with function approximation converges to a region. In *NeurIPS*, pages 1040–1046. MIT Press, 2000.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoffrey Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *ICLR (Poster)*. OpenReview.net, 2018.

Shixiang Gu, Timothy P. Lillicrap, Zoubin Ghahramani, Richard E. Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. In *ICLR*. OpenReview.net, 2017.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 1352–1361. PMLR, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018.

Assaf Hallak and Shie Mannor. Consistent on-line off-policy evaluation. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 1372–1383. PMLR, 2017.

Josiah P. Hanna and Peter Stone. Reducing sampling error in policy gradient learning. In *AAMAS*, pages 1016–1024. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

Alexander Hans, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft. Safe exploration for reinforcement learning. In *ESANN*, pages 143–148, 2008.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2):159–195, 2001.

Jean Harb, Tom Schaul, Doina Precup, and Pierre-Luc Bacon. Policy evaluation networks. *CoRR*, abs/2002.11833, 2020.

Wolfgang Härdle and Léopold Simar. *Applied multivariate statistical analysis*, volume 22007. Springer, 2012.

Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In *NeurIPS*, pages 3315–3323, 2016.

Matthias Heger. Consideration of risk in reinforcement learning. In *ICML*, pages 105–111. Morgan Kaufmann, 1994.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, pages 3215–3222. AAAI Press, 2018.

Timothy Classen Hesterberg. *Advances in importance sampling*. PhD thesis, Citeseer, 1988.

Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2004.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

Tommi S. Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6 (6):1185–1201, 1994a.

Tommi S. Jaakkola, Satinder P. Singh, and Michael I. Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *NeurIPS*, pages 345–352. MIT Press, 1994b.

Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 652–661. JMLR.org, 2016.

Chi Jin, Tiancheng Jin, Haipeng Luo, Suvrit Sra, and Tiancheng Yu. Learning adversarial markov decision processes with bandit feedback and unknown transition. *arXiv preprint arXiv:1912.01192*, 2019.

George H. John. When the best move isn't optimal: Q-learning with exploration. In *AAAI*, page 1464. AAAI Press / The MIT Press, 1994.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NeurIPS*, pages 315–323, 2013.

Sham M. Kakade. A natural policy gradient. In *NeurIPS*, pages 1531–1538. MIT Press, 2001a.

Sham M. Kakade. Optimizing average reward using discounted rewards. In *COLT/EuroCOLT*, volume 2111 of *Lecture Notes in Computer Science*, pages 605–615. Springer, 2001b.

Sham M. Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, pages 267–274. Morgan Kaufmann, 2002.

Abbas Kazerouni, Mohammad Ghavamzadeh, Yasin Abbasi, and Benjamin Van Roy. Conservative contextual linear bandits. In *NeurIPS*, pages 3910–3919, 2017.

Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. *Mach. Learn.*, 49(2-3):209–232, 2002.

Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *ICRA*, pages 8248–8254. IEEE, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

W. Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: the TAMER framework. In *K-CAP*, pages 9–16. ACM, 2009.

Jens Kober and Jan Peters. Policy search for motor primitives in robotics. In *NeurIPS*, pages 849–856. Curran Associates, Inc., 2008.

Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *Int. J. Robotics Res.*, 32(11):1238–1274, 2013.

Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1008–1014. MIT Press, 2000. URL `http://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf`.

Jakub Konecný and Peter Richtárik. Semi-stochastic gradient descent methods. *CoRR*, abs/1312.1666, 2013.

Jakub Konecný, Jie Liu, Peter Richtárik, and Martin Takác. Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE J. Sel. Top. Signal Process.*, 10(2):242–255, 2016.

Augustine Kong. A note on importance sampling using standardized weights. *University of Chicago, Dept. of Statistics, Tech. Rep*, 348, 1992.

Sumit Kunnumkal and Huseyin Topaloglu. Using stochastic approximation methods to compute optimal base-stock levels in inventory control problems. *Oper. Res.*, 56(3):646–664, 2008.

Cornelius Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

Romain Laroche, Paul Trichelair, and Remi Tachet des Combes. Safe policy improvement with baseline bootstrapping. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 3652–3661. PMLR, 2019.

Nicolas Le Roux, Mark Schmidt, and Francis R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NeurIPS*, pages 2672–2680, 2012.

Lihua Lei and Michael I. Jordan. Less than a single pass: Stochastically controlled stochastic gradient. In *AISTATS*, volume 54 of *Proceedings of Machine Learning Research*, pages 148–156. PMLR, 2017.

Lihua Lei, Cheng Ju, Jianbo Chen, and Michael I. Jordan. Nonconvex finite-sum optimization via SCSG methods. *CoRR*, abs/1706.09156, 2017.

Amarildo Likmeta, Alberto Maria Metelli, Andrea Tirinzoni, Riccardo Giol, Marcello Restelli, and Danilo Romano. Combining reinforcement learning with rule-based controllers for transparent and general decision-making in autonomous driving. *Robotics Auton. Syst.*, 131:103568, 2020.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016.

Boyi Liu, Qi Cai, Zhuoran Yang, and Zhaoran Wang. Neural trust region/proximal policy optimization attains globally optimal policy. In *NeurIPS*, pages 10564–10575, 2019a.

Hao Liu, Yihao Feng, Yi Mao, Dengyong Zhou, Jian Peng, and Qiang Liu. Action-dependent control variates for policy optimization via stein identity. In *ICLR (Poster)*. OpenReview.net, 2018a.

Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. In *NeurIPS*, pages 5361–5371, 2018b.

Yao Liu, Pierre-Luc Bacon, and Emma Brunskill. Understanding the curse of horizon in off-policy evaluation via conditional importance sampling. *CoRR*, abs/1910.06508, 2019b.

Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Off-policy policy gradient with state distribution correction. *CoRR*, abs/1904.08473, 2019c.

Daoming Lyu, Qi Qi, Mohammad Ghavamzadeh, Hengshuai Yao, Tianbao Yang, and Bo Liu. Variance-reduced off-policy memory-efficient policy search. *CoRR*, abs/2009.06548, 2020.

Julien Mairal. Incremental majorization-minimization optimization with application to large-scale machine learning. *SIAM J. Optim.*, 25(2):829–855, 2015.

James Martens and Roger B. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2408–2417. JMLR.org, 2015.

Luca Martino, Víctor Elvira, and Francisco Louzada. Effective sample size for importance sampling based on discrepancy measures. *Signal Process.*, 131:386–401, 2017.

Karl Mason and Santiago Grijalva. A review of reinforcement learning for autonomous building energy management. *Comput. Electr. Eng.*, 78:300–312, 2019.

Luca Mastrangelo. A study on off-line policy gradient algorithms and their application to risk-averse learning. 2015.

Takamitsu Matsubara, Tetsuro Morimura, and Jun Morimoto. Adaptive step-size policy gradients with average reward metric. In *ACML*, volume 13 of *JMLR Proceedings*, pages 285–298. JMLR.org, 2010.

Andreas Matthias. The responsibility gap: Ascribing responsibility for the actions of learning automata. *Ethics and information technology*, 6(3):175–183, 2004.

Jincheng Mei, Chenjun Xiao, Csaba Szepesvári, and Dale Schuurmans. On the global convergence rates of softmax policy gradient methods. *CoRR*, abs/2005.06392, 2020.

Alberto Maria Metelli, Matteo Papini, Francesco Faccio, and Marcello Restelli. Policy optimization via importance sampling. In *NeurIPS*, pages 5447–5459, 2018.

Alberto Maria Metelli, Guglielmo Manneschi, and Marcello Restelli. Policy space identification in configurable environments. *CoRR*, abs/1909.03984, 2019.

Alberto Maria Metelli, Flavio Mazzolini, Lorenzo Bisi, Luca Sabbioni, and Marcello Restelli. Control frequency adaptation via action persistence in batch reinforcement learning. *CoRR*, abs/2002.06836, 2020a.

Alberto Maria Metelli, Matteo Papini, Nico Montali, and Marcello Restelli. Importance sampling techniques for policy optimization. *Journal of Machine Learning Research*, 21(141):1–75, 2020b. URL `http://jmlr.org/papers/v21/20-124.html`.

Atsushi Miyamae, Yuichi Nagata, Isao Ono, and Shigenobu Kobayashi. Natural policy gradient methods with parameter-based exploration for control tasks. In *NeurIPS*, pages 1660–1668. Curran Associates, Inc., 2010.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*. OpenReview.net, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016.

Teodor Mihai Moldovan and Pieter Abbeel. Risk aversion in markov decision processes via near optimal chernoff bounds. In *NeurIPS*, pages 3140–3148, 2012a.

Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. In *ICML*. icml.cc / Omnipress, 2012b.

John E. Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE Trans. Neural Networks*, 12(4):875–889, 2001.

Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

Brett L. Moore, Larry D. Pyeatt, Vivekanand Kulkarni, Periklis Panousis, Kevin Padrez, and Anthony G. Doufas. Reinforcement learning for closed-loop propofol anesthesia: a study in human volunteers. *J. Mach. Learn. Res.*, 15(1):655–696, 2014.

Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. Nonparametric return distribution approximation for reinforcement learning. In *ICML*, pages 799–806. Omnipress, 2010.

Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *Int. J. Robotics Res.*, 32 (3):263–279, 2013.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning. In *NeurIPS*, pages 1046–1054, 2016.

Katta G. Murty and Santosh N. Kabadi. Some np-complete problems in quadratic and nonlinear programming. *Math. Program.*, 39(2):117–129, 1987.

Kimia Nadjahi, Romain Laroche, and Rémi Tachet des Combes. Safe policy improvement with soft baseline bootstrapping. In *ECML/PKDD (3)*, volume 11908 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2019.

Yurii E. Nesterov. *Introductory Lectures on Convex Optimization - A Basic Course*, volume 87 of *Applied Optimization*. Springer, 2004.

Gergely Neu, Anders Jonsson, and Vicenç Gómez. A unified view of entropy-regularized markov decision processes. *CoRR*, abs/1705.07798, 2017.

Gerhard Neumann. Variational inference for policy search in changing situations. In *ICML*, pages 817–824. Omnipress, 2011.

Andrew Y. Ng and Michael I. Jordan. PEGASUS: A policy search method for large mdps and pomdps. In *UAI*, pages 406–415. Morgan Kaufmann, 2000.

Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *ICML*, pages 663–670. Morgan Kaufmann, 2000.

Lam M. Nguyen, Jie Liu, Katya Scheinberg, and Martin Takác. SARAH: A novel method for machine learning problems using stochastic recursive gradient. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 2613–2621. PMLR, 2017.

Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cogn. Process.*, 12(4):319–340, 2011.

Chris Nota and Philip S. Thomas. Is the policy gradient a gradient? In *AAMAS*, pages 939–947. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019.

Art B Owen. Monte carlo theory, methods and examples. *Monte Carlo Theory, Methods and Examples. Art Owen*, 2013.

Joni Pajarinen, Hong Linh Thai, Riad Akrour, Jan Peters, and Gerhard Neumann. Compatible natural gradient policy search. *Mach. Learn.*, 108(8-9):1443–1466, 2019.

Matteo Papini, Matteo Pirotta, and Marcello Restelli. Adaptive batch size for safe policy gradients. In *NeurIPS*, pages 3591–3600, 2017.

Matteo Papini, Damiano Binaghi, Giuseppe Canonaco, Matteo Pirotta, and Marcello Restelli. Stochastic variance-reduced policy gradient. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 4023–4032. PMLR, 2018.

Matteo Papini, Alberto Maria Metelli, Lorenzo Lupo, and Marcello Restelli. Optimistic policy optimization via multiple importance sampling. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 4989–4999. PMLR, 2019a.

Matteo Papini, Matteo Pirotta, and Marcello Restelli. Smoothing policies and safe policy gradients. *arXiv preprint arXiv:1905.03231*, 2019b.

Matteo Papini, Andrea Battistello, and Marcello Restelli. Balancing learning speed and stability in policy gradient via adaptive exploration. In *AISTATS*, volume 108 of *Proceedings of Machine Learning Research*, pages 1188–1199. PMLR, 2020.

Martin Pecka and Tomás Svoboda. Safe exploration techniques for reinforcement learning - an overview. In *MESAS*, volume 8906 of *Lecture Notes in Computer Science*, pages 357–375. Springer, 2014.

Jing Peng and Ronald J. Williams. Incremental multi-step q-learning. In *ICML*, pages 226–232. Morgan Kaufmann, 1994.

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *CoRR*, abs/2004.00784, 2020a.

Zilun Peng, Ahmed Touati, Pascal Vincent, and Doina Precup. SVRG for policy evaluation with fewer gradient evaluations. In *IJCAI*, pages 2697–2703. ijcai.org, 2020b.

Jan Peters. Policy gradient methods. *Scholarpedia*, 5(11):3698, 2010.

Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *ICML*, volume 227 of *ACM International Conference Proceeding Series*, pages 745–750. ACM, 2007.

Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9): 1180–1190, 2008a.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008b.

Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. 2003.

Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In *AAAI*. AAAI Press, 2010.

Nhan H. Pham, Lam M. Nguyen, Dzung T. Phan, Phuong Ha Nguyen, Marten van Dijk, and Quoc Tran-Dinh. A hybrid stochastic policy gradient algorithm for reinforcement learning. In *AISTATS*, volume 108 of *Proceedings of Machine Learning Research*, pages 374–385. PMLR, 2020.

Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Adaptive step-size for policy gradient methods. In *NeurIPS*, pages 1394–1402, 2013a.

Matteo Pirotta, Marcello Restelli, Alessio Pecorino, and Daniele Calandriello. Safe policy iteration. In *ICML (3)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 307–315. JMLR.org, 2013b.

Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Policy gradient in lipschitz markov decision processes. *Mach. Learn.*, 100(2-3):255–283, 2015.

Hélène Plisnier, Denis Steckelmacher, Tim Brys, Diederik M. Roijers, and Ann Nowé. Directed policy gradient for safe reinforcement learning with human advice. *CoRR*, abs/1808.04096, 2018.

Hélène Plisnier, Denis Steckelmacher, Diederik M. Roijers, and Ann Nowé. The actor-advisor: Policy gradient with off-policy advice. *CoRR*, abs/1902.02556, 2019.

L. A. Prashanth and Mohammad Ghavamzadeh. Actor-critic algorithms for risk-sensitive reinforcement learning. *CoRR*, abs/1403.6530, 2014.

L. A. Prashanth and Mohammad Ghavamzadeh. Variance-constrained actor-critic algorithms for discounted and average reward mdps. *Mach. Learn.*, 105(3): 367–417, 2016.

Doina Precup, Richard S. Sutton, and Satinder P. Singh. Eligibility traces for off-policy policy evaluation. In *ICML*, pages 759–766. Morgan Kaufmann, 2000.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2005.

Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham M. Kakade. Towards generalization and simplicity in continuous control. In *NeurIPS*, pages 6550–6561, 2017.

Ingo Rechenberg. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*, pages 83–114. Springer, 1978.

Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *CoRR*, abs/1806.09460, 2018.

Sashank J. Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczos, and Alexander J. Smola. Stochastic variance reduction for nonconvex optimization. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 314–323. JMLR.org, 2016.

Alfréd Rényi et al. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California, 1961.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Aviv Rosenberg and Yishay Mansour. Online convex optimization in adversarial markov decision processes. In *ICML*, 2019.

Mark Rowland, Anna Harutyunyan, Hado van Hasselt, Diana Borsa, Tom Schaul, Rémi Munos, and Will Dabney. Conditional importance sampling for off-policy learning. In *AISTATS*, volume 108 of *Proceedings of Machine Learning Research*, pages 45–55. PMLR, 2020.

Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach, Third International Edition*. Pearson Education, 2010.

Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *CoRR*, abs/1703.03864, 2017.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICLR (Poster)*, 2016.

Bruno Scherrer and Matthieu Geist. Local policy search in a convex space and conservative policy iteration as boosted policy search. In *ECML/PKDD (3)*, volume 8726 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2014.

Nicol N Schraudolph. Local gain adaptation in stochastic gradient descent. 1999.

Nicol N. Schraudolph, Douglas Aberdeen, and Jin Yu. Fast online policy gradient learning with SMD gain vector adaptation. In *NeurIPS*, pages 1185–1192, 2005.

John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1889–1897. JMLR.org, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Pieter Abbeel, and Xi Chen. Equivalence between policy gradients and soft q-learning. *CoRR*, abs/1704.06440, 2017a.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017b.

Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Policy gradients with parameter-based exploration for control. In *ICANN (1)*, volume 5163 of *Lecture Notes in Computer Science*, pages 387–396. Springer, 2008.

Lior Shani, Yonathan Efroni, and Shie Mannor. Adaptive trust region policy optimization: Global convergence and faster rates for regularized mdps. In *AAAI*, pages 5668–5675. AAAI Press, 2020.

Yun Shen, Ruihong Huang, Chang Yan, and Klaus Obermayer. Risk-averse reinforcement learning for algorithmic trading. In *CIFEr*, pages 391–398. IEEE, 2014.

Zebang Shen, Alejandro Ribeiro, Hamed Hassani, Hui Qian, and Chao Mi. Hessian aided policy gradient. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 5729–5738. PMLR, 2019.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin A. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 387–395. JMLR.org, 2014.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359, 2017.

Thiago D. Simão and Matthijs T. J. Spaan. Safe policy improvement with baseline bootstrapping in factored environments. In *AAAI*, pages 4967–4974. AAAI Press, 2019.

Satinder P. Singh, Tommi S. Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *ICML*, pages 284–292. Morgan Kaufmann, 1994.

Yi Sun, Daan Wierstra, Tom Schaul, and Jürgen Schmidhuber. Efficient natural evolution strategies. In *GECCO*, pages 539–546. ACM, 2009.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3:9–44, 1988.

Richard S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *AAAI*, pages 171–176. AAAI Press / The MIT Press, 1992.

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

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, pages 1057–1063. The MIT Press, 1999.

Richard S. Sutton, Csaba Szepesvári, and Hamid Reza Maei. A convergent o(n) temporal-difference algorithm for off-policy learning with linear function approximation. In *NeurIPS*, pages 1609–1616. Curran Associates, Inc., 2008.

Richard S. Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *ICML*, volume 382 of *ACM International Conference Proceeding Series*, pages 993–1000. ACM, 2009.

Richard S. Sutton, Ashique Rupam Mahmood, and Martha White. An emphatic approach to the problem of off-policy temporal-difference learning. *J. Mach. Learn. Res.*, 17:73:1–73:29, 2016.

Istvan Szita and András Lörincz. Learning tetris using the noisy cross-entropy method. *Neural Comput.*, 18(12):2936–2941, 2006.

Nassim Nicholas Taleb. *The black swan: The impact of the highly improbable*, volume 2. Random house, 2007.

Aviv Tamar and Shie Mannor. Variance adjusted actor critic algorithms. *CoRR*, abs/1310.3697, 2013.

Aviv Tamar, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor. Sequential decision making with coherent risk. *IEEE Trans. Autom. Control.*, 62(7):3323–3338, 2017.

Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization. In *ICLR (Poster)*. OpenReview.net, 2019.

Devinder Thapa, In-Sung Jung, and Gi-Nam Wang. Agent based decision support system using reinforcement learning under emergency circumstances. In *ICNC (1)*, volume 3610 of *Lecture Notes in Computer Science*, pages 888–892. Springer, 2005.

Philip S. Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2139–2148. JMLR.org, 2016.

Philip S. Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence policy improvement. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2380–2388. JMLR.org, 2015a.

Philip S. Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High-confidence off-policy evaluation. In *AAAI*, pages 3000–3006. AAAI Press, 2015b.

Philip S Thomas, Bruno Castro da Silva, Andrew G Barto, Stephen Giguere, Yuriy Brun, and Emma Brunskill. Preventing undesirable behavior of intelligent machines. *Science*, 366(6468):999–1004, 2019.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

John N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Mach. Learn.*, 16(3):185–202, 1994.

John N. Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *NeurIPS*, pages 1075–1081. MIT Press, 1996.

George Tucker, Surya Bhupatiraju, Shixiang Gu, Richard E. Turner, Zoubin Ghahramani, and Sergey Levine. The mirage of action-dependent baselines in reinforcement learning. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5022–5031. PMLR, 2018.

Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration in finite markov decision processes with gaussian processes. In *NeurIPS*, pages 4305–4313, 2016.

Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration for interactive machine learning. In *NeurIPS*, pages 2887–2897, 2019.

Tim van Erven and Peter Harremoës. Rényi divergence and kullback-leibler divergence. *IEEE Trans. Inf. Theory*, 60(7):3797–3820, 2014.

Hado van Hasselt. Double q-learning. In *NeurIPS*, pages 2613–2621. Curran Associates, Inc., 2010.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100. AAAI Press, 2016.

Harm van Seijen, Hado van Hasselt, Shimon Whiteson, and Marco A. Wiering. A theoretical and empirical analysis of expected sarsa. In *ADPRL*, pages 177–184. IEEE, 2009.

Harm van Seijen, Ashique Rupam Mahmood, Patrick M. Pilarski, Marlos C. Machado, and Richard S. Sutton. True online temporal-difference learning. *J. Mach. Learn. Res.*, 17:145:1–145:40, 2016.

Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *SIGGRAPH*, pages 419–428. ACM, 1995.

Vivek Veeriah, Shangtong Zhang, and Richard S. Sutton. Crossprop: Learning representations by stochastic meta-gradient descent in neural networks. In *ECML/PKDD (1)*, volume 10534 of *Lecture Notes in Computer Science*, pages 445–459. Springer, 2017.

Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Deep conservative policy iteration. In *AAAI*, pages 6070–6077. AAAI Press, 2020.

Nikos Vlassis and Marc Toussaint. Model-free reinforcement learning as mixture learning. In *ICML*, volume 382 of *ACM International Conference Proceeding Series*, pages 1081–1088. ACM, 2009.

Paul Wagner. A reinterpretation of the policy oscillation phenomenon in approximate policy iteration. In *NeurIPS*, pages 2573–2581, 2011.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016a.

Lingxiao Wang, Qi Cai, Zhuoran Yang, and Zhaoran Wang. Neural policy gradient methods: Global optimality and rates of convergence. In *ICLR*. OpenReview.net, 2020.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1995–2003. JMLR.org, 2016b.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. In *ICLR (Poster)*. OpenReview.net, 2017.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Mach. Learn.*, 8: 279–292, 1992.

Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256, 1992.

Cathy Wu, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M. Bayen, Sham M. Kakade, Igor Mordatch, and Pieter Abbeel. Variance reduction for policy gradient with action-dependent factorized baselines. *CoRR*, abs/1803.07246, 2018.

Yifan Wu, Roshan Shariff, Tor Lattimore, and Csaba Szepesvári. Conservative bandits. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1254–1262. JMLR.org, 2016.

Yue Wu, Weitong Zhang, Pan Xu, and Quanquan Gu. A finite time analysis of two time-scale actor critic methods. *CoRR*, abs/2005.01350, 2020.

Yuhuai Wu, Elman Mansimov, Roger B. Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *NeurIPS*, pages 5279–5288, 2017.

Pan Xu, Felicia Gao, and Quanquan Gu. An improved convergence analysis of stochastic variance-reduced policy gradient. In *UAI*, page 191. AUAI Press, 2019.

Pan Xu, Felicia Gao, and Quanquan Gu. Sample efficient policy gradient methods with recursive variance reduction. In *ICLR*. OpenReview.net, 2020.

Tianbing Xu, Qiang Liu, and Jian Peng. Stochastic variance reduction for policy gradient estimation. *CoRR*, abs/1710.06034, 2017.

Zhongwen Xu, Hado van Hasselt, and David Silver. Meta-gradient reinforcement learning. In *NeurIPS*, pages 2402–2413, 2018.

Zhuoran Yang, Yongxin Chen, Mingyi Hong, and Zhaoran Wang. On the global convergence of actor-critic: A case for linear quadratic regulator with ergodic cost. *CoRR*, abs/1907.06246, 2019.

Huizhen Yu. On convergence of some gradient-based temporal-differences algorithms for off-policy learning. *CoRR*, abs/1712.09652, 2017.

Junyu Zhang, Chengzhuo Ni, Zheng Yu, Csaba Szepesvari, and Mengdi Wang. On the convergence and sample efficiency of variance-reduced policy gradient method, 2021.

Junzi Zhang, Jongho Kim, Brendan O'Donoghue, and Stephen P. Boyd. Sample efficient reinforcement learning with REINFORCE. *CoRR*, abs/2010.11364, 2020a.

Shangtong Zhang, Bo Liu, and Shimon Whiteson. Per-step reward: A new perspective for risk-averse reinforcement learning. *CoRR*, abs/2004.10888, 2020b.

Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. Analysis and improvement of policy gradient estimation. In *NeurIPS*, pages 262–270, 2011.

Tingting Zhao, Hirotaka Hachiya, Voot Tangkaratt, Jun Morimoto, and Masashi Sugiyama. Efficient sample reuse in policy gradients with parameter-based exploration. *Neural Comput.*, 25(6):1512–1547, 2013.

Dongruo Zhou, Pan Xu, and Quanquan Gu. Stochastic nested variance reduction for nonconvex optimization. *CoRR*, abs/1806.07811, 2018.

Taiyu Zhu, Kezhi Li, Lei Kuang, Pau Herrero, and Pantelis Georgiou. An insulin bolus advisor for type 1 diabetes using deep reinforcement learning. *Sensors*, 20 (18):5058, 2020.

Alexander Zimin and Gergely Neu. Online learning in episodic markovian decision processes by relative entropy policy search. In *NeurIPS*, pages 1583–1591, 2013.