POLITECNICO DI MILANO

Facoltà di Ingegneria

Scuola di Ingegneria Industriale e dell'Informazione

Dipartimento di Elettronica, Informazione e Bioingegneria

Master of Science in

Computer Science and Engineering – Ingegneria Informatica

Adaptive Batch Size for Safe Policy Gradient Methods

Supervisor:

MARCELLO RESTELLI

Assistant Supervisor:

MATTEO PIROTTA

Master Graduation Thesis by:

MATTEO PAPINI

Student Id n. 850421

Academic Year 2016-2017

To all who shared this journey with me.

RINGRAZIAMENTI (ACKNOWLEDGMENTS)

Ringrazio innanzitutto il Prof. Marcello Restelli, che mi ha guidato nella stesura di questa tesi e nel lavoro di ricerca su cui si basa. La sua disponibilità, insieme alla volontà e alla capacità di trasmettere i frutti della sua esperienza sono stati fondamentali nel conseguimento di questo risultato.

Altrettanto importante è stato l'aiuto di Matteo Pirotta. Nonostante i limiti imposti dalla distanza fisica, i suoi consigli e le sue critiche hanno spinto la ricerca verso risultati più interessanti e il presente lavoro verso una maggiore qualità.

Vorrei ringraziare, naturalmente, i miei genitori, per il sostegno economico, senza il quale non avrei potuto dedicarmi a tempo pieno allo studio, e a quello morale, insostituibile.

Infine un grazie va ai miei amici, la cui compagnia in questi mesi ha contribuito a tenere alto lo spirito, senza il quale nessuna opera può essere portata a termine.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

MDP	Markov Decision Process
RL	Reinforcement Learning
MC	Monte Carlo
TD	Temporal Difference
EM	Expectation Maximization
REPS	Relative Entropy Policy Search
CPI	Conservative Policy Iteration
USPI	Unique-parameter Safe Policy Improvement
MSPI	Multiple-parameter Safe Policy Improvement
LPI	Linearized Policy Improvement
TRPO	Trust Region Policy Optimization
LQG	Linear-Quadratic Gaussian regulation

ABSTRACT

Policy gradient methods are among the best Reinforcement Learning techniques to solve complex control problems. The application of these methods to real systems call for algorithms capable of guaranteeing a constant improvement of the policies that are tried during the learning process, known as safe algorithms. The research on safe policy gradient methods has so far focused on the selection of the step size of policy updates. Another important parameter is the batch size, that is the number of samples used to estimate the gradient direction for each update, but it has not received the same attention so far. In this thesis we propose a method to jointly optimize the step size and the batch size to achieve (with high probability) monotonic improvement. Theoretical guarantees are accompanied by numerical simulations to analyze the behavior of the proposed algorithms.

SOMMARIO

I metodi 'policy gradient' ('gradiente della politica') rappresentano una delle tecniche più efficaci per risolvere problemi di controllo complessi. L'applicazione di questi metodi a sistemi reali rende necessario lo sviluppo di algoritmi cosiddetti 'safe', traducibile con 'prudenti', capaci di garantire un miglioramento costante delle politiche che vengono provate nel corso dell'apprendimento. La ricerca su metodi policy gradient safe si è concentrata, finora, sulla scelta della 'step size', che regola l'entità degli aggiornamenti della politica. Un altro parametro importante è la 'batch size', il numero di campioni usati per stimare la direzione del gradiente per ogni aggiornamento, che finora, tuttavia, non ha ricevuto pari attenzioni. In questa tesi viene proposto un metodo per ottimizzare congiuntamente entrambi i parametri per ottenere (con probabilità elevata) un miglioramento costante della politica. I risultati teorici sono accompagnati da simulazioni numeriche volte ad analizzare il comportamento degli algoritmi proposti.

1 INTRODUCTION

Reinforcement Learning (RL) is a field of machine learning that aims at building intelligent machines capable of learning complex tasks from experience. Among the current challenges that Reinforcement Learning has to face, there are the inner difficulty of learning in continuous domains and the problem of safe learning. The aim of this thesis is to develop a novel approach to safe learning in continuous domains, in the form of an adaptive policy gradient algorithm.

Continuous domains are common in automatic control and robotic applications. Traditional RL algorithms, based on value functions, suffer from many problems when applied to such continuous tasks, the most serious being the lack of strong theoretical guarantees. Policy gradient methods, besides guaranteeing convergence to locally optimal policies [32], are able to address many of the difficulties that characterize complex control problems, such as high dimensional state spaces and noisy sensors. Moreover, they allow to easily incorporate prior domain knowledge in order to design safer and more effective policies. Many different policy gradient algorithms have been proposed, all sharing the same basic principle of computing the gradient of a performance measure to update the agent's policy in a direction of improvement. These methods have been successfully applied to complex control tasks (see [8] for a recent survey). Policy gradient is often used to improve an existing policy, which may be designed by a human expert or learned by simulation. Improvement on the real system can account for the lack of accuracy that necessarily characterizes any model of the environment. Moreover, when the latter is non-stationary, self improvement can make an agent more autonomous, by making it able to adapt to the changes that may occur, without the need of human intervention. We will address the problem of safe learning with this scenario in mind.

During the learning process, it is common to observe oscillations in the agent's performance. In approaching the optimal policy, not all the updates, if taken individually, yield a performance improvement. Although often what counts is the final result, this kind of behavior is unacceptable for many relevant applications. Safe approaches, also known as conservative, aim at guaranteeing monotonic performance improvement during the learning process. There are many reasons for which this property may be desirable, especially in the scenario of our interest. First of all, when learning happens on a real system, time is limited. This means that the promise of a good asymptotic performance is no longer enough, and also partial results become impor-

tant. Steady improvement is even more fundamental when learning is performed simultaneously with production, for instance to adapt to small changes in the environment in a real-time fashion. In this case, worsening updates may directly affect the performance. Finally, particularly bad updates may result in dangerous behaviors able to damage the system.

Safe approaches to RL come from the approximate policy iteration literature, starting from the seminal work by Kakade and Langford [13]. On the basis of their approach, many safe methods have been proposed in the recent years, including safe policy gradient methods. So far, research in this direction has focused on the selection of the step size, also known as learning rate, a meta-parameter that is usually associated with convergence speed, but can also be responsible of performance oscillation. Pirotta et al. [25] devised an adaptive step size that can be used to guarantee monotonic improvement with high probability. Another parameter with similar properties is the batch size, the number of sample trajectories used to estimate the policy gradient. However, to the best of our knowledge, the automatic selection of the batch size has not been taken into consideration so far in the safe RL literature. Large values of the batch size produce more accurate estimates, which result in a steadier improvement. On the other hand, employing a large number of samples is costly, especially in real control applications. Moreover, the selection of the step size is dependent from the batch size that is employed. For all these reasons, the next natural step in safe policy gradient methods is an algorithm where also the batch size is adaptive.

The aim of this work is to develop an adaptive policy gradient algorithm able to automatically select both meta-parameters, the step size and the batch size. Although the focus is on safety, convergence speed can never be neglected in a practical RL algorithm. This means that our algorithm must guarantee monotonic improvement with high probability, but also favor fast convergence when possible. The starting point is represented by the results on adaptive step size for Gaussian policies provided in [25]. We follow fundamentally the same approach, consisting in maximizing a pessimistic lower bound on the performance improvement. The existing results are first generalized, obtaining an even better performance improvement guarantee. The new formulation is then used to include also the batch size in the optimization process. Since the batch size affects convergence speed only in an indirect manner, a new cost function must be defined that takes into account the cost of collecting samples. Theoretical results point out an interesting relationship between the two meta-parameters, and the resulting algorithm is indeed able to guarantee monotonic improvement. Since the method is highly conservative, we propose some empirical variants that allow to keep the selected batch size under

reasonable values. The proposed methods are tested on simple, simulated control task and compared with traditional approaches.

The structure of the remaining part of this document is as follows: Chapter 2 provides preliminary knowledge on Markov Decision Processes and Reinforcement Learning algorithms, with a focus on policy gradient methods. Chapter 3 describes the state of the art of safe approaches to Reinforcement Learning, including safe policy gradient. Chapter 4 contains all the novel theoretical results on the adaptive batch size and presents the new algorithm with all its variants. Chapter 5 discusses the results of numerical simulations on simple control tasks. Chapter 6 draws conclusions on this work and discusses possible future developments. Appendix A provides all the proofs that are omitted in Chapter 4. Finally, Appendix B represents a first attempt at extending the results to other classes of policies.

2 REINFORCEMENT LEARNING PRELIMINARIES

In this chapter we provide an introduction to the reinforcement learning framework and to policy gradient. The aim is to introduce concepts that will be used extensively in the following chapters. For a complete treatment of reinforcement learning, refer to [31].

In Section 2.1 we present the Markov Decision Process model and define the Reinforcement Learning problem. In Section 2.2 we provide an overview of Reinforcement Learning algorithms, while Section 2.3 is entirely dedicated to policy gradient methods.

2.1 MARKOV DECISION PROCESSES

In this section we provide a brief treatment on the model at the root of most RL algorithms, including policy gradient methods: the Markov Decision Process (MDP). Following the opposite direction than the one usually adopted by Reinforcement Learning textbooks, we first formalize continuous MDPs. The discrete MDP model will be introduced later as a special case. This is justified by the scope of our work.

2.1.1 *The model*

Markov decision processes model the very general situation in which an agent interacts with a stochastic, partially observable environment in order to reach a goal. At each time step t, the agent receives observations from the environment. From the current observations and previous interactions, the agent builds a representation of the current state of the environment, s_t. Then, depending on s_t, it takes action a_t. Finally, it is given a (possibly negative) reward depending on how much its behavior is compliant with the goal, in the form of a scalar signal r_{t+1}. The dynamics of the environment, as represented by the agent, must satisfy two fundamental properties:

- Stationarity: the dynamics of the environment does not change over time.
- Markov property: s_{t+1} must depend only on s_t and a_t.

The dynamics can still be stochastic. Under this framework, the goal of the agent can be restated as the problem of collecting as much reward as possible. For now, and every time it is not specified otherwise, we will refer to continuing (i.e. never-ending) tasks, in which future rewards are exponentially discounted.

Formally, a discrete-time continuous Markov decision process is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu \rangle$, where:

- $\mathcal{S} \subseteq \mathbb{R}^n$ is an n-dimensional continuous state space. Elements of this set are the possible states of the environment as observed by the agent.
- $\mathcal{A} \subseteq \mathbb{R}^m$ is an $m-$dimensional continuous action space, from which the agent can pick its actions.
- $\mathcal{P}\colon \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{S})$ is a Markovian transition model, where $\mathcal{P}(s' \mid s, a)$ defines the transition density between states s and s' under action a, i.e. the probability distribution over the next state s' when the current state is s and the chosen action is a. In general, we denote with $\Delta(X)$ the set of probability distributions over X.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to [-R, R]$ is the reward function, such that $\mathcal{R}(s, a)$ is the expected value of the reward received by taking action a in state s and $R \in \mathbb{R}^+$ is the maximum absolute-value reward.
- $\gamma \in [0, 1)$ is the discount factor for future rewards.
- $\mu \in \Delta(\mathcal{S})$ is the initial state distribution, from which the initial state is drawn.

The behavior of the agent is modeled with a stationary, possibly stochastic policy:

$$\pi\colon \mathcal{S} \mapsto \Delta(\mathcal{A}),$$

such that $\pi(s)$ is the probability distribution over the action a to take in state s. We denote with $\pi(a \mid s)$ the probability density of action a under state s. An agent is said to follow policy π if it draws actions from such conditional distribution. We say that a policy π is deterministic if for each $s \in \mathcal{S}$ there exists some $a \in \mathcal{A}$ such that $\pi(a \mid s) = 1$. With abuse of notation, we denote with $\pi(s)$ the action selected with probability 1 by a deterministic policy π in state s.

2.1.2 *Reinforcement Learning: problem formulation*

Given an MDP, the goal of Reinforcement Learning is to find an optimal policy π^* without an a-priori model of the environment, i.e. without knowing $\mathcal{P}$, $\mathcal{R}$ and μ. The optimal policy must be learned by direct interaction with the environment (which may be a real system or a simulation). The objective of the agent is to maximize the total discounted reward, called return v:

$$v = \sum_{t=0}^{\infty} \gamma^t r_t.$$

To evaluate how good a policy π is, we need a measure of performance. A common choice is the expected value of the return over π and the initial state distribution μ:

$$J_\mu^\pi = \mathbf{E}_{\mu,\pi}[v],$$

which we will sometimes call simply 'performance'. It is convenient to introduce a new distribution, called discounted stationary distribution or discounted future-state distribution d_μ^π [32]:

$$d_\mu^\pi(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \Pr(s_t = s \mid \pi, \mu),$$

such that $d_\mu^\pi(s)$ is the probability of having $s_t = s$, discounted with γ^t, in the limit $t \to \infty$. The stationary distribution allows to better define the expected return J as:

$$J_\mu^\pi = \int_{\mathcal{S}} d_\mu^\pi(s)\int_{\mathcal{A}} \pi(a \mid s)\mathcal{R}(s,a)\mathrm{d}a\mathrm{d}s.$$

By introducing the joint distribution $\zeta_{\mu,\pi}$ between d_μ^π and π, we can express the expected return even more compactly:

$$J_\mu^\pi = \mathop{\mathbf{E}}_{(s,a)\sim\zeta_{\mu,\pi}}[\mathcal{R}(s,a)]$$

The RL problem can then be formulated as a stochastic optimization problem:

$$\pi^* \in \arg\max_\pi J(\pi),$$

where π^* is the optimal policy. It is well known that when the space of possible policies is unconstrained, every MDP admits an optimal policy [39].

2.1.3 *Value functions*

Value functions provide a measure of how valuable a state, or a state-action pair, is under a policy π. They are powerful theoretical tools and are employed in many practical RL algorithms. The state value function $V^\pi\colon \mathcal{S} \mapsto \mathbb{R}$ assigns to each state the expected return that is obtained by following π starting from state s, and can be defined recursively by the following equation:

$$V^\pi(s) = \int_{\mathcal{A}} \pi(a|s)\left(\mathcal{R}(s,a) + \gamma\int_{\mathcal{S}} \mathcal{P}(s'|s,a)V^\pi(s')\mathrm{d}s'\right)\mathrm{d}a,$$

which is known as Bellman's expectation equation. The expected return $J_\mu(\pi)$ can be expressed in terms of the state-value function as:

$$J_\mu^\pi = \int_{\mathcal{S}} \mu(s)V^\pi(s)\mathrm{d}s.$$

An optimal policy maximizes the value function in each state simultaneously, i.e. $\pi^* \in \arg\max_\pi V^\pi(s)\ \forall s \in \mathcal{S}$.

For control purposes, it is more useful to define an action-value function $Q^\pi \colon \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, such that $Q^\pi(s, a)$ is the return obtained starting from state s, taking action a and following policy π from then on. Also the action-value function is defined by a Bellman equation:

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

Finally, the difference between the two value functions is known as advantage function:

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

Intuitively, $A(s, a)$ represents the advantage of taking action a in state s instead of drawing an action from the policy. The advantage function allows to define a useful measure of policy improvement. The policy advantage of policy π' w.r.t. π under initial state distribution μ is:

$$\mathbb{A}_{\pi,\mu}(\pi') = \int_{\mathcal{S}} d_\mu^\pi(s) \int_{\mathcal{A}} \pi'(a \mid s) A^\pi(s, a) da ds.$$

Intuitively, $\mathbb{A}_{\pi,\mu}(\pi')$ measures how much π' selects advantageous actions w.r.t. π.

2.2 OVERVIEW OF REINFORCEMENT LEARNING ALGORITHMS

In this section we provide an overview of the main classes of RL algorithms, highlighting the differences between the policy iteration approach and the policy search approach.

2.2.1 *Partial taxonomy*

Reinforcement learning algorithms can be classified according to many different criteria.

First, it is useful to discriminate between the target policy, that is the output of the learning algorithm, and behavior policies, which are policies used to interact with the environment. This produces a first distinction:

- On-policy algorithms: the behavior policy coincides with the target policy.
- Off-policy algorithms: a target policy is learned independently from the adopted behavior policies.

If we consider the degree of interleaving between learning and experience, we have the following classification (which is independent on whether experience is real or simulated):

- Continuing algorithms: learning is on-line. The target policy is updated possibly at each time step.
- Episodic algorithms: experience is divided into finite segments, called episodes. The target policy is updated in between episodes.
- Batch algorithms: Learning ad experience are separated. The target policy is learned by looking at data previously collected with a set of behavior policies.

Most of the RL algorithms that we will consider are on-policy and episodic. An episode of length H under policy π is typically obtained by starting from a state drawn from μ, following π and stopping after H time steps. We call trajectory the sequence $\langle s_t, a_t, r_t \rangle_{t=0,\dots,H}$, where each reward r_t was obtained by taking action a_t in state s_t.

Another useful distinction is the following:

- Model-based algorithms: the agent has an explicit model of the environment.
- Model-free algorithms: the agent has no model of the environment.

When the model of the environment is exact, the RL problem can be solved by using dynamic programming algorithms. A more interesting situation is when the environment dynamics must be estimated from data. However, we will mainly consider model-free algorithms.

Finally, most RL methods can be classified into the following two classes:

- Value function methods rely on the computation of a value function to learn better and better policies.
- Direct policy search methods directly search for an optimal solution in the space of policies.

Most value-function based algorithms fall into the policy iteration family. Policy iteration is actually a dynamic programming algorithm, but most model-free value-function based methods follow the general scheme of policy iteration, hence the name. We will briefly discuss policy iteration in the following. Policy gradient is a direct policy search method, and is thoroughly described in Section 2.3.

2.2.2 *Policy Iteration*

Policy iteration is typically applied to discrete MDPs. Discrete MDPs can be defined as a special case of MDPs, as defined in the previous section, where both the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are discrete, finite sets. All the definitions introduced so far can be adapted to the discrete case simply by replacing integrals with sums.

We report the general policy iteration scheme. Starting from an initial policy, $\pi \leftarrow \pi_0$, policy iteration alternates between the two following phases:

1. Policy evaluation: Given the current policy π, compute (or estimate) action-value function Q^π.
2. Policy improvement: Compute a better policy π' based on Q^π, then set $\pi \leftarrow \pi'$ and go to 1.

A typical policy update for phase 2 is the greedy policy improvement:

$$\pi'(s) \in \arg\max_{a \in \mathcal{A}} Q^\pi(s, a),$$

which employs deterministic policies. When the value function Q^π can be computed exactly for each state-action pair, policy iteration with greedy policy improvement is guaranteed to converge to an optimal policy [27].

In most RL problems the value function Q^π cannot be computed exactly, but can be estimated from samples. Given the sampling nature of the approach, some stochasticity must be included in the policy to ensure sufficient exploration of the state-action space. Several algorithms are known for discrete MDPs with reasonably small state and action spaces [31]. These algorithms are also called tabular, because the value function can be stored in a finite table. Tabular algorithms mainly differ for the way the value function is estimated. The two main families are Monte Carlo (MC) methods, which are episodic, and Temporal Difference (TD) algorithms, which are continuing. Convergence guarantees are available for most of these algorithms [36] [30] [7] [38] [11].

Unfortunately, tabular algorithms cannot be applied to the more general case of continuous MDPs. One possibility is to discretize the state and action spaces. This is possible, but not trivial, and may not be suitable for most tasks. Moreover, tabular methods are unfeasible even for discrete MDPs when the cardinality of the state-action space is too high. Another possibility is to replace the value function with a function approximator. The supervised learning literature provides very powerful function approximators, such as neural networks, which are able to approximate functions of arbitrary complexity starting from a finite parametrization. However, the convergence properties of policy iteration methods employing function approximators are not well understood. The problem is that two levels of approximation are used: the function approximator tries to map action-state pairs to returns, which in turn are sampled from limited experience, creating a sort of moving target situation. Although function approximation has been successfully applied to complex RL problems, its

lack of theoretical guarantees makes it unfeasible for many relevant applications. These problems are discussed in more depth in Section 2.3.1.

2.2.3 *Policy search*

Contrary to value function approaches, direct policy search consists in exploring a subset of the policy space directly to find an optimal policy. The fact that it does not require to estimate the value function makes this approach suitable for continuous MDPs, such as the one arising in robotics. For a recent survey on policy search methods refer to [8]. The following section is entirely dedicated to policy gradient. Other notable model-free policy search methods are Expectation Maximization (EM) algorithms and Relative Entropy Policy Search (REPS).

2.3 POLICY GRADIENT

In this section we will focus on a direct policy search approach with good theoretical properties and promising practical results: policy gradient.

2.3.1 *Why policy gradient*

As mentioned in Section 2.2.2, policy iteration requires to employ function approximators to deal with continuous MDPs. We now present in more detail the main problems of this approach:

- Function approximation methods have no guarantee of convergence, not even to locally optimal policies, and there are even examples of divergence. Besides compromising the learning process, in some applications such instabilities may damage the system.
- Greedy policies based on approximated value functions can be highly sensitive to small perturbations in the state, making them unfeasible for tasks characterized by noise, e.g. control tasks with noisy sensors.
- Given the complexity of the value function representation (think of a multi-layer perceptron), it is difficult to isolate undesired behaviors, such as potentially dangerous actions.

Policy gradient methods bypass all these problems by searching directly for the optimal policy. Convergence to a local optimum is guaranteed [16], uncertainty in the state have more controllable effects, policies can be made safe by design and prior domain knowledge can be exploited to design ad-hoc policies for specific tasks. Moreover, ap-

proximate value functions can still be used to guide the search for the optimal policy, such as in actor-critic methods.

Indeed, policy gradient methods (and other direct policy search approaches) have been successfully applied to complex control tasks. In [29] the authors were able to solve a robot standing task, where a (simulated) biped robot must keep an erect position while perturbed by external forces. In [14] Kober and Peters used policy search in combination with imitation learning to teach the Ball-in-a-Cup game to a real robotic arm. A similar approach was used in [19] for a baseball swing task. In [18] policy search was used to play simulated robot table tennis.

2.3.2 *Definition*

Policy gradient is a kind of policy search based on gradient descent, in which the search is restricted to a class of parameterized policies:

$$\Pi_\theta = \{\pi_\theta : \theta \ \in \boldsymbol{\Theta} \subseteq \mathbb{R}^m\},$$

where θ is the parameter vector. The parametric policy π_θ can have any shape, but is required to be differentiable in θ. Like all gradient descent methods, the parameter vector is updated in the direction of the gradient of a performance measure, which is guaranteed to be the direction of maximum improvement. In our case the performance measure is the expected return $J_\mu^{\pi_\theta}$, which we overload as $J_\mu(\theta)$ for the sake of readability. The gradient of the expected return w.r.t. the parameter vector, $\nabla_\theta J_\mu(\theta)$, is called policy gradient. Starting from an arbitrary initial parametrization θ_0, policy gradient methods perform updates of the kind:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J_\mu(\theta),$$

where $\alpha \geqslant 0$ is the learning rate, also called step size. The update is performed at each learning iteration. Convergence to at least a local optimum is guaranteed by the gradient descent update.

2.3.3 *Policy Gradient Theorem and eligibility vectors*

The Policy Gradient Theorem is a result by Sutton et al. [32] that relates the policy gradient to the value function Q^π:

Theorem 2.3.1 (Continuous version of Theorem 1 from [32])**.** *For any MDP,*

$$\nabla_\theta J_\mu(\theta) = \int_S d_\mu^\pi(s) \int_A \nabla_\theta \pi_\theta(a \mid s) Q^\pi(s,a) da\, ds.$$

The Policy Gradient Theorem is of key importance for many policy gradient algorithms.

2.3.4 *Stochastic gradient descent*

In many practical tasks it is impossible, or unfeasible, to compute the exact policy gradient $\nabla_\theta J_\mu(\theta)$, but a gradient estimate $\hat{\nabla}_\theta J_\mu(\theta)$ can be estimated from sample trajectories. The gradient estimate is often averaged over a number N of such trajectories, also called a batch. The size N of the batch is called batch size. The stochastic gradient descent update is:

$$\theta \leftarrow \theta + \alpha \hat{\nabla}_\theta J_\mu(\theta),$$

where, at each learning iteration, N trajectories need to be performed. Convergence to a local optimum is still guaranteed, provided that the angular difference between $\nabla_\theta J_\mu(\theta)$ and $\hat{\nabla}_\theta J_\mu(\theta)$ is less than $\pi/2$.

2.3.5 *A general policy gradient algorithm*

We provide here the skeleton of a general policy gradient algorithm:

Algorithm 2.1 A general policy gradient algorithm

set initial policy parametrization θ
while not converged **do**
  perform N trajectories following policy π_θ
  compute policy gradient estimate $\hat{\nabla}_\theta J_\mu(\theta)$
  $\theta \leftarrow \theta + \alpha \hat{\nabla}_\theta J_\mu(\theta)$

Many elements are left open in this schema: how to pick batch size N and step size α, what class of policies to use, how to compute the gradient estimate and how to devise a practical stopping condition. In the following sections we will see actual policy gradient algorithms, all adding details or introducing small variations to this general schema.

2.3.6 *Finite differences*

Among the oldest policy gradient methods we find finite-difference methods [9], which arose in the stochastic simulation literature. The idea is to perturb the policy parameter θ many times, obtaining $\theta + \Delta\theta_1 \dots \theta + \Delta\theta_K$, and estimate for each perturbation the expected return difference $\Delta\hat{J}_i \simeq J(\theta + \Delta\theta_i) - J(\theta)$, by performing a number of sample trajectories. After collecting all these data, the policy gradient can be estimated by regression as:

$$\hat{\nabla}_\theta J_\mu(\theta) = (\boldsymbol{\Delta\Theta}^\top \boldsymbol{\Delta\Theta})^{-1} \boldsymbol{\Delta\Theta}^\top \boldsymbol{\Delta\hat{J}}$$

where $\boldsymbol{\Delta\Theta}^\top = [\Delta\theta_1, \dots, \Delta\theta_K]^\top$ and $\boldsymbol{\Delta\hat{J}} = [\Delta\hat{J}_1, \dots, \Delta\hat{J}_K]^\top$. This method is easy to implement and very efficient when applied to deterministic tasks or pseudo-random number simulations. In real control tasks

though, perturbing the policy parametrization without incurring in instability may not be trivial and noise can have a severe impact on performance. Moreover, this kind of gradient estimation requires a large number of trajectories.

2.3.7 *Likelihood Ratio*

The likelihood ratio approach allows to estimate the gradient even from a single trajectory, without the need of perturbing the parametrization. This method is due to Williams [40], but is better understood from the perspective of the later Policy Gradient Theorem [32]. By applying trivial differentiation rules ($\nabla \log f = \frac{\nabla f}{f}$) to the expression of the policy gradient (see Theorem 2.3.1), we have:

$$\nabla_\theta J_\mu(\theta) = \int_{\mathcal{S}} d_\mu^\pi(s) \int_{\mathcal{A}} \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a) \mathrm{d}a \mathrm{d}s.$$

This is known as the REINFORCE [40] trick. The term

$$\nabla_\theta \log \pi_\theta(a \mid s) = \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$$

has many names: eligibility vector, characteristic eligibility, score function and likelihood ratio. In terms of the joint distribution $\zeta_{\mu,\theta}$, the policy gradient is just:

$$\nabla_\theta J_\mu(\theta) = \mathop{\mathbf{E}}_{(s,a)\sim\zeta_{\mu,\theta}} \left[\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a)\right].$$

Since sampling state-action pairs from ζ is equivalent to following policy π_θ, the gradient can be estimated from a single trajectory as:

$$\tilde{\nabla}_\theta J_\mu(\theta) = \langle \nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a) \rangle_H,$$

where $\langle \cdot \rangle_H$ denotes sample average over H time steps. Of course a more stable estimate can be obtained by averaging again over a batch of N trajectories:

$$\hat{\nabla}_\theta J_\mu(\theta) = \langle \langle \nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s,a) \rangle_H \rangle_N.$$

2.3.8 *Baselines*

A problem of the REINFORCE approach is the large variance of the estimate, which results in slower convergence. The Policy Gradient Theorem still holds when an arbitrary baseline $b(s)$ is subtracted from the action-value function, as shown in [40]:

$$\nabla_\theta J_\mu(\theta) = \mathop{\mathbf{E}}_{(s,a)\sim\zeta} \left[\nabla_\theta \log \pi_\theta(a \mid s) (Q^\pi(s,a) - b(s))\right].$$

The baseline can be chosen to reduce the variance of the gradient estimate without introducing any bias, speeding up convergence. A

natural choice of baseline is the state-value function V^π, because it normalizes the value of the actions according to the overall value of the state, preventing overestimation of the possible improvement. This is equivalent to employ the advantage function in place of the action-value function:

$$\begin{aligned}\nabla_\theta J_\mu(\theta) &= \mathop{\mathbf{E}}_{(s,a)\sim\zeta}[\nabla_\theta \log \pi_\theta(a \mid s)(Q^\pi(s,a) - V^\pi(s))] \\ &= \mathop{\mathbf{E}}_{(s,a)\sim\zeta}[\nabla_\theta \log \pi_\theta(a \mid s)(A^\pi(s,a))]\,.\end{aligned}$$

The usage of the advantage function with the eligibility vector has an intuitive meaning: to follow the policy gradient direction means to assign to advantageous actions a higher probability of being taken.

2.3.9 *The REINFORCE algorithm*

In practical tasks, the exact value function Q^π is not available and must be estimated. Episodic likelihood ratio algorithms estimate it from the total return at the end of the episode, assigning to each visited state-action pair a value in retrospective. The REINFORCE gradient estimator [40] uses the total return directly:

$$\hat{\nabla}_\theta J_\mu^{RF}(\theta) = \left\langle \sum_{k=1}^{H} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \left(\sum_{l=1}^{H} \gamma^{l-1} r_l^n - b \right) \right\rangle_N .$$

A problem of the REINFORCE algorithm is the high variance of the gradient estimate, which can be reduced by using a proper baseline. A variance-minimizing baseline can be found in [20]:

$$b = \frac{\left\langle \left(\sum_{k=1}^{H} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \right)^2 \sum_{l=1}^{H} \gamma^{l-1} r_l^n \right\rangle_N}{\left\langle \left(\sum_{k=1}^{H} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \right)^2 \right\rangle_N}.$$

Note that, in the case $|\theta| > 1$, all operations among vectors are to be intended per-component or, equivalently, each gradient component $\hat{\nabla}_{\theta_i} J_\mu(\theta)$ must be computed separately.

2.3.10 *The PGT/G(PO)MDP algorithm*

A problem of the REINFORCE gradient estimator is that the value estimate for (s_k^n, a_k^n) depends on past rewards. Two refinements are the PGT gradient estimator [33]:

$$\hat{\nabla}_\theta J_\mu^{PGT}(\theta) = \left\langle \sum_{k=1}^{H} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \left(\sum_{l=k}^{H} \gamma^{l-1} r_l^n - b \right) \right\rangle_N ,$$

and the G(PO)MDP gradient estimator [4]:

$$\hat{\nabla}_\theta J_\mu^{G(PO)MDP}(\theta) = \left\langle \sum_{l=1}^{H} \left(\sum_{k=1}^{l} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \right) (\gamma^{l-1} r_l^n - b_l) \right\rangle_N .$$

Although the algorithms are different, the gradient estimator that is computed is exactly the same, i.e. $\hat{\nabla}_\theta J_\mu^{PGT}(\theta) = \hat{\nabla}_\theta J_\mu^{G(PO)MDP}(\theta)$, as shown in [20]. From now on we will refer to the PGT form. In the case $b = 0$, the PGT gradient estimation has been proven to suffer from less variance than REINFORCE [41] under mild assumptions. A variance-minimizing baseline for PGT is provided in [20]. Differently from the REINFORCE case, a separate baseline b_l is used for each time step l:

$$b_l = \frac{\left\langle \left(\sum_{k=1}^{l} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \right)^2 \gamma^{l-1} r_l^n \right\rangle_N}{\left\langle \left(\sum_{k=1}^{l} \nabla_\theta \log \pi_\theta(a_k^n \mid s_k^n) \right)^2 \right\rangle_N}.$$

2.3.11 *Natural gradients*

The algorithms described so far use the so-called vanilla policy gradient, i.e., an unbiased estimate of the true gradient w.r.t. policy parameters. Since such algorithms perform a search in parameter space Θ, the performance strongly depends on the policy parametrization. Natural policy gradients allow to search directly in policy space, independently on the parametrization. The natural policy gradient update rule is [12]:

$$\theta \leftarrow \theta + F_\theta^{-1} \nabla_\theta J_\mu(\theta),$$

where F_θ is the Fisher information matrix:

$$\begin{aligned} F_\theta &= \int_S d_\mu^\pi(s) \int_A \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s)^T da ds \\ &\simeq \langle \nabla_\theta \log \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s)^T \rangle_H. \end{aligned}$$

The parameter update is in the direction of a rotated policy gradient. Since the angular difference is never more than $\pi/2$ [2], convergence to a local optimum is still guaranteed. Although a good policy parametrization can yield better performance for specific problems, natural policy gradients offer a more general approach and have shown effective in practice [19].

2.3.12 *Actor-critic methods*

Actor-critic methods try to combine policy search and value function estimation to perform low-variance gradient updates. The "actor" is

a parametrized policy π_θ, which can be updated as in vanilla policy gradient algorithms. For this reason, the policy gradient methods described so far are also called actor-only. The "critic" is an estimate of the action-value function $\hat{Q}_w$, where w is a parameter vector. The approximate value-function methods mentioned in Section 2.2.2 can be thought as critic-only. The RL literature provides a great number of methods to learn a value function estimate from experience [31]. We provide an example of episodic value-function parameter update:

$$w \leftarrow w + \beta\langle(R_t - \hat{Q}_w(s_t, a_t))\nabla_w \hat{Q}_w(s_t, a_t)\rangle_H,$$

where $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ and β is just a learning rate. Actor and critic meet when $\hat{Q}_w$ is used to compute the policy gradient used to update θ:

$$\theta \leftarrow \theta + \langle\nabla_\theta \log \pi_\theta(a \mid s)\hat{Q}_w(s, a)\rangle_H. \tag{2.1}$$

This approach exploits the predictive power of approximate value functions while preserving the nice theoretical properties of the policy gradient. Under some assumptions on the critic function (see Theorem 2 in [32]), the Policy Gradient Theorem is still valid when $\hat{Q}_w$ is used in place of the true value function Q^π:

$$\nabla_\theta J_\mu(\theta) = \int_S d_\mu^\pi(s) \int_A \pi_\theta(a \mid s)\nabla_\theta \log \pi_\theta(a \mid s)\hat{Q}_w(s, a)da\,ds.$$

This means that parameter update (2.1) is still in the direction of the policy gradient, guaranteeing convergence to a local optimum. Additionally, an approximator of the value function V^π can be used as a baseline to reduce variance. For a recent survey of actor-critic methods refer to [10], which reports also examples of the usage of natural gradients in combination with the actor-critic approach.

2.3.13 *Common policy classes*

Although there are no restrictions on parameterized policies, some specific classes are used extensively in applications and have well studied properties. One of the most popular is the Gaussian policy, which in its most general form is defined as:

$$\pi_\theta(a \mid s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{a - \mu_\theta(s)}{\sigma_\theta(s)}\right)^2\right).$$

A common form is the Gaussian policy with fixed standard deviation σ and mean linear in a feature vector $\boldsymbol{\phi}(s)$:

$$\pi_\theta(a \mid s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{a - \theta^T\boldsymbol{\phi}(s)}{\sigma}\right)^2\right),$$

having eligibility vector:

$$\nabla_\theta \log \pi_\theta(a \mid s) = \frac{(a - \theta^\top \phi(s))\phi(s)}{\sigma^2}.$$

When the action space $\mathcal{A}$ is discrete, a common choice is the softmax policy:

$$\pi_\theta(a \mid s) = \frac{e^{\theta^\top \phi(s,a)}}{\sum_{a' \in \mathcal{A}} e^{\theta^\top \phi(s,a')}},$$

having eligibility vector:

$$\nabla_\theta \log \pi_\theta(a \mid s) = \phi(s,a) - \sum_{a' \in \mathcal{A}} \pi_\theta(s \mid a)\phi(s,a').$$

3

SAFE REINFORCEMENT LEARNING: STATE OF THE ART

The aim of this chapter is to present the state of the art in safe approaches to Reinforcement Learning, also known as conservative or monotonically improving methods.

In Section 3.1 we define the policy oscillation problem and provide the motivations behind safe methods. In Section 3.2 we describe safe approaches to policy iteration. In Section 3.3 we present existing safe policy gradient methods. Finally, in Section 3.4 we mention other interesting safe approaches to RL.

3.1 PROBLEM DEFINITION AND MOTIVATION

3.1.1 *The policy oscillation problem*

Safe RL algorithms address a problem known as policy oscillation, first studied in the context of approximate policy iteration and not yet fully understood (see for instance [5], [37]). Policy oscillation affects approximate RL algorithms and manifests as a non-monotonic performance $J_\mu(\pi)$ over learning iterations. In parametric methods, a consequence is that the convergence of the parameter vector to a stationary point may not correspond to an increasing performance, but to an oscillating one. Even when the trend of $J_\mu(\pi)$ is generally positive, large oscillations can be unacceptable for some applications. Among the main recognized causes of policy oscillation are the entity of approximation errors and the magnitude of policy updates. To explain the phenomenon in more detail, we restrict our analysis to policy gradient methods. However, the nature of the problem is fundamentally the same also for policy iteration methods, as shown by Wagner [37]. For policy gradient methods, non-monotonic improvement means that in some cases $J_\mu(\theta + \alpha\hat{\nabla}_\theta J_\mu(\theta)) \leqslant J_\mu(\theta)$. The two main causes are:

1. The magnitude of the updates, controlled by the step size α. Large parameter updates may result in overshooting in policy space, producing oscillation around a local optimum.
2. The policy gradient estimation error. The variance of $\hat{\nabla}_\theta J_\mu(\theta)$ affects the quality of the updates in terms of direction, and may even produce updates in non-improving directions.

3.1.2 *The need for safe methods*

Both of the policy oscillation causes that have been described pose a trade-off between monotonic improvement and speed of convergence. Using small step sizes results in more conservative policy updates, reducing performance oscillation. On the other hand, a small step size can heavily affect convergence speed. Similarly, computing more accurate policy gradient estimates typically requires a large number of samples. Even if the batch size does not affect directly the number of iterations to convergence, it can affect the actual learning time, especially when collecting sample trajectories is time-wise expensive. The resulting trade-off is similar to the well known exploration-exploitation dilemma, and is particularly evident when learning is performed on real systems. In a simulated task with no time limitations, neither convergence speed nor oscillation are major issues, unless convergence is compromised. Instead, in more realistic scenarios, oscillation itself may cause the following problems:

- Dangerous behavior: low performance peaks may correspond to large deviations from the initial policy. Even if the initial policy has been designed to be safe, such deviations may result in behaviors able to damage the system.
- Poor long-term performance: Consider the scenario in which learning is performed simultaneously with production, e.g. to automatically adjust an existing optimal policy to small changes in the environment, either in a real-time manner or by temporary stopping production. Oscillations in the per-episode performance are paid in terms of long-term performance when they are not justified by a relevant enough overall policy improvement. This loss is similar to the concept of regret caused by exploration.
- Selection of suboptimal policies: if the learning process has a fixed duration or must be stopped for some reason, oscillations may lead to the selection of a policy that is not the best seen so far, or is even worse than the initial policy. In highly stochastic environments it can be very difficult to manually reject such sub-par policies, and testing all the policies seen so far may be prohibitively expensive.

From all these problems comes the necessity of policy updates that are monotonically increasing.

3.2 SAFE POLICY ITERATION

In policy iteration, policy oscillation can arise each time approximation is used either in the evaluation step or in the improvement step.

In this section we present the main safe approaches approximate to policy iteration, starting from the seminal work by Kakade and Langford [13].

3.2.1 *Conservative Policy Iteration*

In [13], Kakade and Langford address the policy oscillation problem by limiting the magnitude of the policy updates. Let π be the current policy and π' a greedy policy w.r.t. the latest action-value estimate, i.e. :

$$\pi'(a \mid s) \in \arg\max_{a \in \mathcal{A}} \hat{Q}^{\pi}(s, a)$$

The traditional approximate policy update is:

$$\pi_{new}(a \mid s) = \pi'(a \mid s).$$

Instead, Kakade and Langford proposed to update π with a convex combination of the greedy and the current policy:

$$\pi_{new}(a \mid s) = (1-\alpha)\pi(a \mid s) + \alpha\pi'(a \mid s), \tag{3.1}$$

with step size $\alpha \in [0, 1]$. When a step size $\alpha < 1$ is used, this is more conservative than a pure greedy improvement, producing a new policy that is more similar to the original one. The authors were able to compute a lower bound on the performance improvement:

$$J_{\mu}(\pi_{new}) - J_{\mu}(\pi) \geqslant \frac{\alpha}{1-\gamma}\left(\mathbb{A}_{\pi,\mu}(\pi') - \frac{2\alpha\gamma\epsilon}{1-\gamma(1-\alpha)}\right),$$

where $\epsilon = \max_{s \in \mathcal{S}} |\mathbf{E}_{a \sim \pi'}[A^{\pi}(s, a)]|$. While the positive term obviously depends on the advantage of the greedy policy, the negative one must be intended as a penalty for large policy updates. By maximizing this bound w.r.t. α, a step size is obtained that guarantees positive performance improvement as long as the advantage is positive, without totally sacrificing the speed of convergence:

$$\alpha^* = \frac{(1-\gamma)\mathbb{A}_{\pi,\mu}(\pi')}{4R} \tag{3.2}$$

$$J_{\mu}(\pi_{new}) - J_{\mu}(\pi) \geqslant \frac{\mathbb{A}^2_{\pi,\mu}(\pi')}{8R} \tag{3.3}$$

When the true advantage cannot be computed and an estimate must be used, monotonic improvement can still be guaranteed with high probability. The resulting algorithm is called Conservative Policy Iteration (CPI). This methodology inspired the development of several later safe methods.

3.2.2 *Unique and Multiple-Parameter Safe Policy Improvement*

Pirotta et al. [26] derived a more general bound on the performance improvement of a policy π' over π:

$$J_\mu(\pi_{new}) - J_\mu(\pi) \geqslant \mathbb{A}_{\pi,\mu}(\pi') - \frac{\gamma}{(1-\gamma)^2} \left\|\pi' - \pi\right\|_\infty^2 \frac{\|Q^\pi\|_\infty}{2}, \quad (3.4)$$

where the penalty induced by large policy updates is made more evident by the term $\|\pi' - \pi\|_\infty^2$. Starting from this bound, a new algorithm called Unique-parameter Safe Policy Improvement (USPI) was developed. USPI uses the same update as CPI (3.1), but with a step size based on a better improvement bound, derived from 3.4. This results in better guaranteed performance improvements. In the same paper, the authors also propose a more general, state dependent policy update:

$$\pi_{new}(a \mid s) = (1 - \alpha(s))\pi(a \mid s) + \alpha(s)\pi'(a \mid s).$$

Using state-dependent step-sizes, it is possible to limit the update to the states in which the average advantage $\mathbf{E}_{a\sim\pi'}[A^\pi(s,a)]$ is positive, by setting $\alpha(s) = 0$ for all other states. This is the idea behind Multiple-parameter Safe Policy Improvement (MSPI). Although more general, this algorithm is not guaranteed to provide better performance improvements than USPI, although it is never worse than CPI. Moreover, the computation of the optimal step-sizes for 'good' states can only be done iteratively, limiting efficiency.

3.2.3 *Linearized Policy Improvement*

Bartlett et al. [1] propose a different policy update employing state-dependent mixture coefficients, which in this case can be computed efficiently. The policy update is:

$$\pi_{new}(a \mid s) = \nu(a \mid s)(1 + \alpha\Delta_\pi(s,a)),$$

where:

- $\nu(\cdot \mid s)$ is a policy such that:

$$\mathop{\mathbf{E}}_{a\sim\pi}\left[\hat{Q}^\pi(s \mid a)\right] = \mathop{\mathbf{E}}_{a\sim\nu}\left[\hat{Q}^\pi(s \mid a)\right] + \mathop{\mathbf{Var}}_{a\sim\nu}\left[\hat{Q}^\pi(s \mid a)\right], \quad (3.5)$$

  which can be computed for each visited state with a binary search.

- $\Delta_\pi(s,a)$ is a normalized action-value function:

$$\Delta_\pi(s,a) = \hat{Q}^\pi(s,a) - \mathop{\mathbf{E}}_{a\sim\nu}\left[\hat{Q}^\pi(s,a)\right], \quad (3.6)$$

- α is a step size.

The idea is the following: the original policy π is reshaped into:

$$\overline{\pi}(a \mid s) = \nu(a \mid s)(1 + \Delta_\pi(s, a)),$$

which has the same value function (in the absence of estimation error), i.e. $V^{\overline{\pi}} = V^\pi$. The negative term in Equation 3.6 ensures that $\overline{\pi}$ is a well defined distribution. Introducing a step size α allows to assign higher probability to actions with positive normalized value function Δ_π. A conservative choice for α is $1/\|\Delta_\pi\|_\infty$, but the authors also propose more practical alternatives. The resulting algorithm is called Linearized Policy Improvement (LPI). The authors show that the performance improvement guarantee of LPI is better than the one of CPI, at the same time allowing for bigger policy updates, hence faster convergence.

3.3 SAFE POLICY GRADIENT

In this section we present the state of the art in safe policy gradient methods and suggest a possible direction for further developments.

3.3.1 *The role of the step size*

Most of the safe policy gradient methods that have been proposed so far focus on the selection of the proper step size α. The step size regulates the magnitude of the parameter updates. Although the relationship between $\Delta\theta$ and the actual change in the policy is parametrization-dependent (at least for vanilla policy gradients) and may be non-trivial, in general small step sizes produce small changes in the policy, hence small performance oscillations. However, reducing the step-size also reduces the convergence speed, which is a major drawback in most applications. The traditional solution is to employ line search to select the optimal step size for each problem. Although effective in supervised learning [16], this approach can be prohibitively expensive in RL. An alternative is to make the step size adaptive, i.e. to compute it at each learning iteration from collected data.

3.3.2 *Adaptive step-size*

Pirotta et al. [24] apply the approach of Kakade and Langford [13] to policy gradient. Starting from a continuous version of Bound 3.4, they derive a series of bounds on performance improvement which can be maximized w.r.t. the step size α. We report here a bound for

Gaussian policies which does not require to compute the action-value function:

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \alpha \|\nabla_\theta J_\mu(\theta)\|_2^2 - \alpha^2 \frac{RM_\phi^2 \|\nabla_\theta J_\mu(\theta)\|_1^2}{(1-\gamma)^2\sigma^2} \left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)} \right), \quad (3.7)$$

where $\theta' = \theta + \alpha\nabla_\theta J_\mu(\theta)$ and M_ϕ is a bound on state features $\phi(s)$. At each learning iteration, given $\nabla_\theta J_\mu(\theta)$, the step size that optimizes this bound can be computed:

$$\alpha^* = \frac{\|\nabla_\theta J_\mu(\theta)\|_2^2 (1-\gamma)^3\sigma^3\sqrt{2\pi}}{RM_\phi^2 \|\nabla_\theta J_\mu(\theta)\|_1^2 (2|\mathcal{A}|(1-\gamma) + \sqrt{2\pi}\sigma\gamma)}.$$

Updating the policy parameters with α^* guarantees a constant performance improvement:

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \frac{\|\nabla_\theta J_\mu(\theta)\|_2^4 (1-\gamma)^3\sigma^3\sqrt{2\pi}}{2RM_\phi^2 \|\nabla_\theta J_\mu(\theta)\|_1^2 (2|\mathcal{A}|(1-\gamma) + \sqrt{2\pi}\sigma\gamma)}.$$

When the policy gradient cannot be computed exactly, monotonicity can be guaranteed with high probability, depending on the entity of the estimation error. The authors provide methods, based on Chebyshev's inequality, to select the number of samples necessary to achieve a desired improvement probability δ, for the cases of REINFORCE and PGT gradient estimators.

The same authors [25] follow a similar approach to derive tighter bounds for the case of Lipschitz-continuous MDPs. A Lipschitz MDP is an MDP with Lipschitz-continuous transition model $\mathcal{P}$ and reward function $\mathcal{R}$. We omit the mathematical details, but intuitively the Lipschitz hypothesis is reasonable when the dynamics of the environment are sufficiently smooth. Starting from this hypothesis, and constraining the policy to be Lipschitz-continuous as well, the authors are able to prove the Lipschitz continuity of the policy gradient $\nabla_\theta J_\mu(\theta)$ and provide an efficient algorithm to compute an optimal step size based on this fact.

3.3.3 *Beyond the step size*

Although the step size is crucial in controlling policy oscillation, we saw that also the entity of the estimation error plays a role. In likelihood ratio methods, the variance of the policy gradient estimate $\hat{\nabla}_\theta J_\mu(\theta)$ can be limited by averaging over a large number of trajectories. In this case, the batch size N can affect policy improvement. In [24] the effect of N is recognized, but not exploited: it is still a meta-parameter that must be selected beforehand by a human expert.

As in the case of the step size, line search algorithm are usually too expensive given the time that is typically required to collect a sample trajectory. For the same reason, an adaptive batch size should take into consideration the cost of collecting more samples, which cannot be neglected in practical applications. In the following chapter we address this problems to derive a cost-sensitive adaptive batch size that guarantees monotonic improvement.

3.4 OTHER SAFE METHODS

In this section we describe other safe approaches to reinforcement learning.

3.4.1 *Trust Region Policy Optimization*

A problem with the methods proposed so far is that they tend to be over conservative, negatively affecting convergence speed. Schulman et al. [28] revisit the CPI approach to develop an effective, although strongly heuristic, safe direct policy search algorithm. Given a parametrized policy π_θ, said θ the current parameter and θ' a generic updated parameter, the following is shown to hold (adapted to our notation):

$$J_\mu(\theta) \geqslant L_\theta(\theta') - \alpha H_{max}(\theta, \theta'),$$

where $L_\theta(\theta') = J_\mu(\theta) + \mathop{\mathbf{E}}_{s\sim d_\mu^{\pi_\theta}, a\sim\pi_{\theta'}}[A_{\pi_\theta}(s,a)]$ is a first order approximation of the performance improvement, $\alpha = \frac{2\max_{s\in\mathcal{S}} \mathbf{E}_{a\sim\pi_{\theta'}}[A^\pi(s,a)]}{(1-\gamma)^2}$ is a penalty coefficient and $H_{max}(\theta,\theta') = \max_{s\in\mathcal{S}} H(\pi_\theta, \pi\theta)$, where $H(\cdot,\cdot)$ is the Kullback-Liebler divergence between two distributions. The traditional approach would be to maximize the bound:

$$\theta' = \arg\max_{\tilde{\theta}\in\Theta} L_\theta(\tilde{\theta}) - \alpha H_{max}(\theta, \tilde{\theta}),$$

but the theoretically justified penalty coefficient α leads to updates that are too conservative. Instead, the authors propose to resolve to the following constrained optimization problem:

$$\theta' = \arg\max_{\tilde{\theta}\in\Theta} L_\theta(\tilde{\theta}) \tag{3.8}$$

$$\text{subject to } H_{max}(\theta, \tilde{\theta}) \leqslant \delta. \tag{3.9}$$

Constraint 3.9 is called trust region constraint and its intuitive role is to limit the update to a neighborhood of the current parameter θ to avoid oscillation. A series of approximations to this exact method lead to the Trust Region Policy Optimization (TRPO) algorithm, which shows to be effective on complex tasks.

3.4.2 *High Confidence Policy Improvement*

The methods seen so far try to achieve monotonic improvement. Thomas et al. [35] adopt a looser definition of safety: an algorithm is safe if it always returns a policy with performance below J_μ^- with probability at most δ, where both J_μ^- and δ are selected by the user. In this way, different applications allow for different safety requirements, depending on risk aversion. The authors propose a batch RL algorithm: the set of sample trajectories is partitioned into a training set and a test set. The training set is searched for the policy that is predicted to achieve the best performance among all the policies that are predicted to be safe. The safety of the candidate policy is then tested on the test set. The authors also describe an episodic variant of this algorithm, called DAEDALUS. If compared with other safe approaches, these algorithms require less data and have no meta-parameters requiring expert tuning, but have a higher computational complexity and do not guarantee monotonic improvement.

3.4.3 *Robust Baseline Regret Minimization*

Petrik et al. [21] propose a model-based batch algorithm that guarantees improvement over a baseline policy π. Given a model of the environment $\hat{P}$ and estimation error e, uncertainty set $\Xi(\hat{P}, e)$ is the set of transition dynamics within e. The new policy is selected by solving the following adversarial optimization problem:

$$\pi' \in \arg\max_{\tilde{\pi}} \min_{\xi \in \Xi} \left(J_\mu(\tilde{\pi} \mid \xi) - J_\mu(\pi \mid \xi) \right), \tag{3.10}$$

which selects the best policy given the worst possible dynamics. Solutions to this problem are guaranteed to be safe. Unfortunately, solving 3.10 is NP-hard. The authors provide an approximate algorithm (Approximate Robust Baseline Regret Minimization) with promising results.

4

ADAPTIVE BATCH SIZE FOR POLICY GRADIENT

In this chapter we propose a new safe approach to policy gradient. As mentioned in Section 3.3.3, the safe policy gradient algorithms that have been proposed so far in the literature have focused on the choice of the optimal step size, while, to the best of our knowledge, the choice of the batch size has been neglected. We propose a method to jointly optimize both meta-parameters in a way that is both safe and cost-sensitive. We focus on the class of Gaussian policies, which is the most commonly used to address control problems with continuous action spaces.

In Section 4.1 we outline the whole procedure. In Section 4.2 we generalize and improve an existing result on the adaptive step size. Building on this results, in Section 4.3 we derive the optimal batch size and provide the jointly optimizing algorithm and some variants.

4.1 OVERVIEW

In this section we provide an overview of the approach that we will follow to obtain the adaptive batch size, justifying the main choices. Although our focus is on the batch size, the step size is still a fundamental parameter in controlling policy oscillation phenomena. For this reason, we start from an optimal step-size formulation to jointly optimize both meta-parameters. To do so, we improve an existing adaptive step-size method [24]. We then find the best batch size to use with such an optimal step size. However, it is not trivial to define a proper objective function for this goal. In the case of the step size, performance improvement $J_\mu(\theta') - J_\mu(\theta)$ is representative both of safety and of convergence speed. The batch size N, instead, affects learning times in an indirect way: larger batches require more time to collect samples, but this is not captured by $J_\mu(\theta') - J_\mu(\theta)$. What we need is an objective that is sensitive to the cost of collecting sample trajectories, i.e., a cost-sensitive objective function. We decide to employ the following one:

$$\Upsilon = \frac{J_\mu(\theta') - J_\mu(\theta)}{N}.$$

This simple objective has a strong and intuitive motivation: maximizing Υ means to maximize the average *per-trajectory* performance improvement. In this way, the employment of more samples must be justified by a sufficient performance improvement. This intuition makes Υ, in our opinion, the most appropriate objective function for the cost-sensitive scenario. Following this approach we derive the

adaptive batch size, finally providing an algorithm that guarantees monotonic improvement with high probability, without resorting to impractically high batch sizes.

4.2 ADAPTIVE COORDINATE DESCENT

In this section we improve the adaptive step-size method for Gaussian policies from Pirotta et al. [24], which have been described in Section 3.3.2. We follow the same approach, with the major variant of adopting a non-scalar step size. This allows to control the update of the single policy parameter components independently, potentially allowing for better optimization. Indeed, we are able to show that this approach yields an improvement over the performance guarantee of [24]. Rather surprisingly, this result is achieved with a coordinate descent update, where only one component at a time is updated. We first stick to the theoretical setting in which the gradient is known exactly, then we take into account also the gradient estimation error.

4.2.1 *Exact Framework*

The idea is to have a separate adaptive step size α_i for each component θ_i of θ. For notational convenience, we define a non-scalar step size as a diagonal matrix:

$$\Lambda = \mathrm{diag}(\alpha_1, \alpha_2, \ldots, \alpha_m),$$

with $\alpha_i \geqslant 0$ for $i = 1, \ldots, m$. The policy parameters can be updated as:

$$\theta' = \theta + \Lambda \nabla_\theta J_\mu(\theta).$$

Notice that the direction of the update can differ from the gradient direction, but, being the α_i non-negative, the absolute angular difference is never more than $\pi/2$, so that convergence to local optima can still be assured. The traditional scalar step-size update can be seen as a special case where $\Lambda = \alpha I$.

By adapting Theorem 4.3 in [24] to the new parameter update, we can obtain a lower bound on the policy performance improvement. To do so, we first need the following assumption on state features:

Assumption 4.2.1. *State features are uniformly bounded:* $|\phi_i(s)| \leqslant M_\phi, \forall s \in \mathcal{S}, \forall i = 1, \ldots, m$.

The bound on state features naturally arises in many applications. Even when this is not the case, it can be imposed in the feature design phase or inferred from observations. We can now state the following:

Lemma 4.2.2. *For any initial state distribution μ and any pair of stationary Gaussian policies $\pi_\theta \sim \mathcal{N}(\theta^T \boldsymbol{\phi}(s), \sigma^2)$ and $\pi_{\theta'} \sim \mathcal{N}(\theta'^T \boldsymbol{\phi}(s), \sigma^2)$, so that $\theta' = \theta + \Lambda \nabla_\theta J_\mu(\theta)$, and under Assumption 4.2.1, the difference between the performance of $\pi_{\theta'}$ and the one of π_θ can be bounded below as follows:*

$$\begin{aligned} J_\mu(\theta') - J_\mu(\theta) \geqslant \nabla_\theta J_\mu(\theta)^T \Lambda \nabla_\theta J_\mu(\theta) \\ - \frac{\|\Lambda \nabla_\theta J_\mu(\theta)\|_1^2 M_\phi^2}{(1-\gamma)\sigma^2} \\ \left(\frac{1}{\sqrt{2\pi}\sigma} \int_S d_\mu^{\pi_\theta}(s) \int_A Q^{\pi_\theta}(s,a) da ds + \frac{\gamma \|Q^{\pi_\theta}\|_\infty}{2(1-\gamma)} \right), \end{aligned}$$

where $\|Q^{\pi_\theta}\|_\infty$ is the supremum norm of the Q-function:

$$\|Q^{\pi_\theta}\|_\infty = \sup_{s \in S, a \in A} Q^{\pi_\theta}(s,a).$$

The above bound requires to compute the Q-function explicitly, but, as already mentioned, this is often not possible in real-world applications. We now consider a simplified (although less tight) version of the bound that does not have this requirement, which is an adaptation of Corollary 5.1 in [24]:

Theorem 4.2.3. *For any initial state distribution μ and any pair of stationary Gaussian policies $\pi_\theta \sim \mathcal{N}(\theta^T \boldsymbol{\phi}(s), \sigma^2)$ and $\pi_{\theta'} \sim \mathcal{N}(\theta'^T \boldsymbol{\phi}(s), \sigma^2)$, so that $\theta' = \theta + \Lambda \nabla_\theta J_\mu(\theta)$, and under Assumption 4.2.1, the difference between the performance of $\pi_{\theta'}$ and the one of π_θ can be bounded below as follows:*

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \nabla_\theta J_\mu(\theta)^T \Lambda \nabla_\theta J_\mu(\theta) - c \|\Lambda \nabla_\theta J_\mu(\theta)\|_1^2,$$

where $c = \frac{R M_\phi^2}{(1-\gamma)^2 \sigma^2} \left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)} \right)$ and $|\mathcal{A}|$ is the volume of the action space.

The proofs of both bounds are provided in Appendix A.

We then find the step size Λ^* that maximizes this lower bound. To do so, we study the bound from Theorem 3.3 as a function of $\alpha_1, \ldots, \alpha_m$ constrained by $\alpha_i \geqslant 0 \ \forall i = 1, \ldots, m$. The function has no stationary points. Moreover, except from degenerate cases, partial maximization is possible only along a single dimension at a time. Candidates for the optimum are thus attained only on the boundary, and are obtained by setting to 0 all the components of the step size but one, say α_k, which is set to $\frac{1}{2c}$. This assignment yields:

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \frac{\nabla_{\theta_k} J_\mu(\theta)^2}{4c}.$$

To obtain the global maximum is enough to select k so as to maximize the above quantity, which results in the following:

Corollary 4.2.4. *The performance lower bound of Theorem 4.2.3 is maximized by the following non-scalar step size:*

$$\alpha_k^* = \begin{cases} \frac{1}{2c} & \text{if } k = \arg\max_i |\nabla_{\theta_i} J_\mu(\theta)|, \\ 0 & \text{otherwise}, \end{cases}$$

which guarantees the following performance improvement:

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \frac{\|\nabla_\theta J_\mu(\theta)\|_\infty^2}{4c}.$$

The performance improvement guarantee is obtained simply by substituting the optimal value for Λ back into the bound of theorem 4.2.3. Notice that the update induced by the optimal Λ corresponds to employing a constant, scalar step size to update just one parameter at a time (the one corresponding to the greatest gradient magnitude). This method is known in the literature as coordinate descent [6]. We propose an intuitive explanation of this result: the bound from Theorem 4.2.3 is part of the family of performance improvement bounds originated from [13]. What has been observed in Section 3.2.1 about the original bound is also valid here: the positive part accounts to the average advantage of the new policy over the old one, while the negative part penalizes large parameter updates, which may result in overshooting. Updating just the parameter corresponding to the larger policy gradient component represents an intuitive trade-off between these two objectives. We now show how this result represents an improvement w.r.t. the adaptive scalar step size proposed in [24] for the current setting:

Corollary 4.2.5. *Under identical hypotheses, the performance improvement guaranteed by Corollary 4.2.4 is never less than the one guaranteed by Corollary 5.1 from [24], i.e.:*

$$\frac{\|\nabla_\theta J_\mu(\theta)\|_\infty^2}{4c} \geqslant \frac{\|\nabla_\theta J_\mu(\theta)\|_2^4}{4c\,\|\nabla_\theta J_\mu(\theta)\|_1^2}.$$

This corollary derives from the trivial norm inequality:

$$\|\nabla_\theta J_\mu(\theta)\|_\infty \|\nabla_\theta J_\mu(\theta)\|_1 \geqslant \|\nabla_\theta J_\mu(\theta)\|_2^2.$$

4.2.2 *Approximate Framework*

We now turn to the more realistic case in which the policy gradient, $\nabla_\theta J_\mu(\theta)$, is not known, and has to be estimated from a finite number of trajectory samples. In this case, a performance improvement can still be guaranteed with high probability. We denote with $\hat{\nabla}_\theta J_\mu(\theta)$ the estimated policy gradient, which can be obtained, for instance, with one of the algorithms described in Section 2.3.7. We denote with

$\epsilon_i \geqslant 0$ a high-probability upper bound on the approximation error of the gradient component $\nabla_{\theta_i} J_\mu(\theta)$, i.e.,:

$$\Pr(|\nabla_{\theta_i} J_\mu(\theta) - \hat{\nabla}_{\theta_i} J_\mu(\theta)| \leqslant \epsilon_i) \geqslant 1 - \delta.$$

The estimation error heavily depends on the batch size, so we leave the problem of computing such bounds to the following section. To adapt the result of Theorem 4.2.3 to the stochastic gradient case, we need both a lower bound and an upper bound on $\hat{\nabla}_\theta J_\mu(\theta)$.

$$\underline{\hat{\nabla}_\theta J_\mu}(\theta) = \max(|\hat{\nabla}_\theta J_\mu(\theta)| - \epsilon, 0)$$
$$\overline{\hat{\nabla}_\theta J_\mu}(\theta) = |\hat{\nabla}_\theta J_\mu(\theta)| + \epsilon$$

where $\epsilon = [\epsilon_1, \ldots, \epsilon_m]$ and the maximum operator is componentwise. We can now state the following:

Theorem 4.2.6. *Under the same assumptions of Theorem 4.2.3, and provided that a policy gradient estimate $\hat{\nabla}_\theta J_\mu(\theta)$ is available, so that $\mathbb{P}(|\nabla_{\theta_i} J_\mu(\theta) - \hat{\nabla}_{\theta_i} J_\mu(\theta)| \geqslant \epsilon_i) \leqslant \delta, \forall i = 1, \ldots, m$, the difference between the performance of $\pi_{\theta'}$ and the one of π_θ can be bounded below with probability at least $(1-\delta)^m$ as follows:*

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \underline{\hat{\nabla}_\theta J_\mu}(\theta)^T \Lambda \underline{\hat{\nabla}_\theta J_\mu}(\theta) - c \left\| \Lambda \overline{\hat{\nabla}_\theta J_\mu}(\theta) \right\|_1^2,$$

where c is defined as in Theorem 4.2.3.

To derive the optimal step size, we need the following assumption:

Assumption 4.2.7. *At least one component of the policy gradient estimate is, in absolute value, no less than the approximation error: $\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty \geqslant \epsilon$.*

The violation of the above assumption can be used as a stopping condition, since it prevents to guarantee any performance improvement. The derivation of the optimal step size is similar to the one of Corollary 4.2.5, and the result is again a coordinate descent algorithm. The only difference is that we must select the parameter to update as follows:

$$\theta_k \mid k = \arg\max_i \frac{\max(|\hat{\nabla}_{\theta_i} J_\mu(\theta)| - \epsilon_i, 0)^2}{4c(|\hat{\nabla}_{\theta_i} J_\mu(\theta)| + \epsilon_i)^2}.$$

We restrict our analysis to the case in which $\epsilon_1 = \epsilon_2 = \ldots = \epsilon_m$. We denote this common estimation error simply as ϵ. This comes natural in the following section, where we use concentration bounds to give an expression for ϵ. However, it is always possible to define a common error by setting $\epsilon \triangleq \max_i \epsilon_i$. This allows to simplify the above criterion as:

$$k = \arg\max_i \frac{\max(|\hat{\nabla}_{\theta_i} J_\mu(\theta)| - \epsilon, 0)^2}{4c(|\hat{\nabla}_{\theta_i} J_\mu(\theta)| + \epsilon)^2}.$$

Then, since we are already maximizing, the $\max(\cdot, 0)$ operator can be removed (under Assumption 4.2.7):

$$k = \arg\max_i \frac{(|\hat{\nabla}_{\theta_i} J_\mu(\theta)| - \epsilon)^2}{4c(|\hat{\nabla}_{\theta_i} J_\mu(\theta)| + \epsilon)^2}.$$

Being the objective function monotonic non-decreasing in $|\hat{\nabla}_{\theta_k} J_\mu(\theta)|$, we can simply select k as to maximize $|\hat{\nabla}_{\theta_k} J_\mu(\theta)|$, obtaining the following:

Corollary 4.2.8. *The performance lower bound of Theorem 4.2.6 is maximized under Assumption 4.2.7 by the following non-scalar step size:*

$$\alpha_k^* = \begin{cases} \frac{(\|\hat{\nabla}_\theta J_\mu(\theta)\|_\infty - \epsilon)^2}{2c(\|\hat{\nabla}_\theta J_\mu(\theta)\|_\infty + \epsilon)^2} & \text{if } k = \arg\max_i |\hat{\nabla}_{\theta_i} J_\mu(\theta)|, \\ 0 & \text{otherwise}, \end{cases}$$

which guarantees with probability $(1-\delta)^m$ *a performance improvement*

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \frac{\left(\|\hat{\nabla}_\theta J_\mu(\theta)\|_\infty - \epsilon\right)^4}{4c\left(\|\hat{\nabla}_\theta J_\mu(\theta)\|_\infty + \epsilon\right)^2}.$$

Again, the performance improvement guarantee is obtained simply by substitution.

4.3 ADAPTIVE BATCH SIZE

In this section we jointly optimize the step size for parameter updates and the batch size for policy gradient estimation, taking into consideration the cost of collecting sample trajectories. We call N the batch size, i.e., the number of trajectories sampled to compute the policy gradient estimate $\hat{\nabla}_\theta J_\mu(\theta)$ at each parameter update. We define the following cost-sensitive performance improvement measure:

Definition 4.1. Cost-sensitive performance improvement measure Υ_δ is defined as:

$$\Upsilon_\delta(\Lambda, N) := \frac{B_\delta(\Lambda, N)}{N},$$

where B_δ is the high probability lower bound on performance improvement of Theorem 4.2.6.

We now show how to jointly select the step size Λ and the batch size N so as to maximize Υ_δ. Notice that the dependence of B_δ on N is entirely through ϵ, which can be computed with a concentration bound. Of course, we would like ϵ to be as tight as possible, but we must also consider the tractability of the resulting optimization problem. We first restrict our analysis to concentration bounds that allow to express ϵ as follows:

Assumption 4.3.1. *The per-component policy gradient estimation error made by averaging over* N *sample trajectories can be bounded with probability at least* $1-\delta$ *by:*

$$\epsilon(N) = \frac{d_\delta}{\sqrt{N}},$$

where d_δ *is a constant w.r.t.* N.

This class of inequalities includes well-known concentration bounds such as Chebyshev's and Hoeffding's. Under Assumption 4.3.1 Υ_δ can be optimized in closed form:

Theorem 4.3.2. *Under the hypotheses of Theorem 4.2.3 and Assumption 4.3.1, the cost-sensitive performance improvement measure* Υ_δ*, as defined in Definition 4.1, is maximized by the following step size and batch size:*

$$\alpha_k^* = \begin{cases} \frac{(13-3\sqrt{17})}{4c} & \text{if } k = \arg\max_i |\hat{\nabla}_{\theta_i} J_\mu(\theta)|, \\ 0 & \text{otherwise}, \end{cases}$$

$$N^* = \left\lceil \frac{(13+3\sqrt{17})d_\delta^2}{2\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty^2} \right\rceil,$$

where $c = \frac{RM_\phi^2}{(1-\gamma)^2\sigma^2}\left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)}\right)$*, which guarantee with probability* $(1-\delta)^m$ *a performance improvement of:*

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \frac{393-95\sqrt{17}}{8}\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty^2 \geqslant 0.16\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty^2.$$

The complete proof is left to Appendix A, but Figure 4.1 provides an intuition. The cost function Υ_δ is plotted over the value of N. Real data from one of the experiments of Section 5.1 were used. The function is enhanced to make the peaks more visible, without altering the overall shape. We can see two stationary points: call N_0 the minimum and N^* the maximum (marked with a diamond). N_0 has value 0 and is also the global minimum. This point corresponds to the case $\epsilon = \left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty$. Going towards $N = 0$, Υ_δ goes to infinity, because of the $1/N$ cost factor. However, for $N < N_0$, Assumption 4.2.7 is not satisfied, so no performance improvement can be guaranteed at all. In the feasible region, $N \geqslant N_0$, local maximum N^* is also the global maximum. This is the optimal batch size.

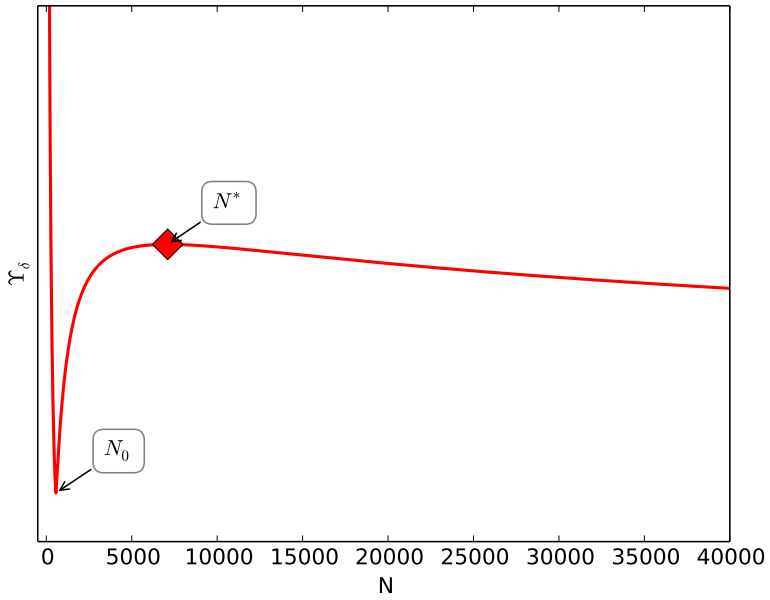


Figure 4.1: Cost function Υ_δ over batch size N. The optimal batch size N^* is marked with a diamond. Batch sizes under the global minimum N_0 do not guarantee any performance improvement.

The main peculiarity of this result is that the step size is constant, in the sense that its value does not depend on the gradient estimate. This can be explained in terms of a duality between step size and batch size: in other conservative adaptive-step size approaches, such as the one proposed with Theorem 4.3.2, the step size is kept small to counteract policy updates that are too off due to bad gradient estimates. When also the batch size is made adaptive, a sufficient number of sample trajectories can be taken to keep the policy update on track even with a constant-valued step size.

We now analyze this approach from a more algorithmic perspective. As already mentioned, under Assumption 4.3.1, Assumption 4.2.7 can be restated as:

$$N \geqslant N_0 \triangleq \frac{d_\delta^2}{\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty^2},$$

which is always verified by the proposed N^*. This means that the adaptive batch size never allows an estimation error larger than the gradient estimate. This is good news, but it may also lead to very large batch sizes once θ is very close to the optimal parameter. An additional stopping condition may be necessary, and one option is simply to set a maximum allowed batch size.

Notice also that the optimal batch size, as defined in Theorem 4.3.2, is referred to the upcoming parameter update. However, the optimal

Algorithm 4.1 Adaptive Policy Gradient

set initial policy parametrization θ
set initial batch size N and maximum batch size $\overline{N}$
$c \leftarrow \frac{RM_\Phi^2}{(1-\gamma)^2\sigma^2}\left(\frac{|A|}{\sqrt{2\pi}\sigma}+\frac{\gamma}{2(1-\gamma)}\right)$
$\epsilon \leftarrow 0$
while $\epsilon < |\hat{\nabla}_\theta J_\mu(\theta)|$ **do**
  perform N trajectories following policy $\pi_\theta \sim \mathcal{N}(\theta^T\boldsymbol{\phi}(s), \sigma)$
  compute policy gradient estimate $\hat{\nabla}_\theta J_\mu(\theta)$
  $k \leftarrow \arg\max_i |\hat{\nabla}_{\theta_i} J_\mu(\theta)|$
  compute d_δ with a proper concentration bound
  $\epsilon \leftarrow d_\delta/\sqrt{N}$
  $\alpha \leftarrow \frac{\left(\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty - \epsilon\right)^2}{2c\left(\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty + \epsilon\right)^2}$
  $\theta_k \leftarrow \theta_k + \alpha\hat{\nabla}_{\theta_k} J_\mu(\theta)$
  $N \leftarrow \left\lceil \frac{(13+3\sqrt{17})d_\delta^2}{2\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty^2} \right\rceil$
  $N \leftarrow \max\{2, \min\{N, \overline{N}\}\}$

batch size can be employed only for the next update, since trajectory samples need to be collected first. This means that, in any practical implementation, the adaptive batch size that we have defined is always "a learning iteration too late". However, it is reasonable to assume that the conditions that determined the optimal batch size will not vary significantly between a learning iteration and the following one. While we need to rely on N's predictive power, we can keep the step size synchronized as follows: compute Λ^* as in Corollary 4.2.8, using:

$$\epsilon = \frac{d_\delta}{\sqrt{N}},$$

where d_δ is computed with the latest available data and N is the batch size used to collect them. A collateral effect of this approach is that, due to integer approximation, the step size is no longer constant. However, the variations in its value should be quite small.

These expedients lead to Algorithm 4.1. Notice that there is still a parameter that must be selected by an expert: δ. This is the probability of having a worsening update, since monotonic improvement is guaranteed with probability $(1-\delta)^m$. Intuitively, the choice of δ depends on risk attitude and should be small in critical applications. However, numerical simulations (see Chapter 5) show that, at least for simple problems, δ can be set to very high values (~ 1) without compromising monotonic improvement. This can be useful, since larger values of δ lead to faster convergence.

We now consider some concentration bounds in more detail, providing the values for d_δ and the corresponding explicit expressions for N^*.

4.3.1 *Chebyshev's bound*

We report here the sample mean version of Chebyshev's bound for a generic stochastic variable X:

$$\Pr(|\mathbb{E}[X] - \overline{X}_N| \geqslant \epsilon) \leqslant \frac{\mathrm{Var}[X]}{N},$$

where $\overline{X}_N$ is the sample mean over N samples. In our case we have:

$$\delta = \Pr(|\nabla_{\theta_i} J_\mu(\theta) - \hat{\nabla}_{\theta_i} J_\mu(\theta)| \geqslant \epsilon) \leqslant \frac{\mathrm{Var}[\hat{\nabla}_{\theta_i} J_\mu(\theta)]}{N}.$$

This bound is compliant with Assumption 4.3.1 and gives the following expression for d_δ:

$$d_\delta = \sqrt{\frac{\mathrm{Var}[\tilde{\nabla}_{\theta_i} J_\mu(\theta)]}{\delta}},$$

where $\tilde{\nabla}_{\theta_i} J_\mu(\theta)$ is the policy gradient approximator (from a single sample trajectory). The main advantage of this bound is that it does not make any assumption on the range of the gradient sample.

Using results from [42] as adapted in [24], we can give explicit formulations for the optimal batch size. When the REINFORCE [40] gradient estimator ($\hat{\nabla}_\theta J_\mu^{RF}(\theta)$) is used to estimate the gradient, we can use Lemma 5.4 from [24] to bound the estimation error as:

$$\epsilon \leqslant \frac{1}{\sqrt{N}} \left(\frac{R M_\phi (1-\gamma^H)}{\sigma(1-\gamma)} \sqrt{\frac{H}{\delta}} \right),$$

which gives an optimal batch size:

$$N^* = \frac{(13 + 3\sqrt{17}) R^2 M_\phi^2 H (1-\gamma^H)^2}{2\delta\sigma^2(1-\gamma)^2 \left\| \hat{\nabla}_\theta J_\mu^{RF}(\theta) \right\|_\infty^2}.$$

Similarly, when the G(PO)MDP/PGT gradient estimator ($\hat{\nabla}_\theta J_\mu^{PGT}(\theta)$) is used, using Lemma 5.5 from [24] we have:

$$\epsilon \leqslant \frac{1}{\sqrt{N}} \left(\frac{R M_\phi}{\sigma(1-\gamma)} \sqrt{\frac{1}{\delta} \left[\frac{1-\gamma^{2H}}{1-\gamma^2} + H\gamma^{2H} - 2\gamma^H \frac{1-\gamma^H}{1-\gamma} \right]} \right)$$

and

$$N^* = \frac{(13 + 3\sqrt{17}) R^2 M_\phi^2 \left[\frac{1-\gamma^{2H}}{1-\gamma^2} + H\gamma^{2H} - 2\gamma^H \frac{1-\gamma^H}{1-\gamma} \right]}{2\delta\sigma^2(1-\gamma)^2 \left\| \hat{\nabla}_\theta J_\mu^{PGT}(\theta) \right\|_\infty^2}.$$

These results are independent from the baseline used in the gradient estimation. The G(PO)MDP/PGT estimator suffers from a smaller variance if compared with REINFORCE, and the variance bound is indeed tighter.

4.3.2 *Hoeffding's bound*

We report Hoeffding's bound applied to our case:

$$\delta = \Pr(|\nabla_{\theta_i} J_\mu(\theta) - \hat{\nabla}_{\theta_i} J_\mu(\theta)| \geqslant \epsilon) \leqslant 2e^{-\frac{2\epsilon^2}{NR^2}},$$

where $\mathbf{R}$ is the range of the gradient approximator, i.e. $|\text{supp}(\tilde{\nabla}_{\theta_i} J_\mu(\theta))|$. This bound is compliant with Assumption 4.3.1 and gives:

$$d_\delta = \mathbf{R}\sqrt{\frac{\log 2/\delta}{2}},$$

For the class of policies we are considering, i.e. Gaussian with mean linear in the features, the range can be upper bounded under some assumptions:

Lemma 4.3.3. *For any Gaussian policy* $\pi_\theta \sim \mathcal{N}(\theta^T \boldsymbol{\phi}(s), \sigma^2)$*, assuming that the action space is bounded* ($|a| \leqslant \overline{A}\ \forall a \in \mathcal{A}$) *and the policy gradient is estimated on trajectories of length* H*, the range* $\mathbf{R}$ *of the policy gradient sample* $\tilde{\nabla}_{\theta_i} J_\mu(\theta)$ *can be upper bounded* $\forall i = 1, \ldots, m$ *and* $\forall \theta$ *by*

$$\mathbf{R} \leqslant \frac{2HM_\phi \overline{A} R}{\sigma^2(1-\gamma)}.$$

As we will show in Chapter 5, a more practical solution (even if less rigorous) consists in computing the range as the difference between the largest and the smallest gradient sample seen during learning.

4.3.3 *Empirical Bernstein's bound*

Tighter concentration bounds allow for smaller batch sizes (which result in more frequent policy updates) and larger step sizes, thus speeding up the learning process and improving long-time average performance. An empirical Bernstein bound from [15] allows to use sample variance instead of the variance bounds from [42] and to limit the impact of the gradient range. We report the bound for a generic stochastic variable X:

$$|E[X] - \overline{X}_N| \leqslant \sqrt{\frac{2S_N \ln 3/\delta}{N}} + \frac{3\mathbf{R} \ln 3/\delta}{N},$$

where S_N is the sample variance of X, defined as:

$$S_N = \frac{1}{N}\sum_{i=1}^{N}(X_i - \overline{X}_N)^2.$$

Unfortunately, this bound does not satisfy Assumption 4.3.1, giving for the estimation error the following, more complex, expression:

$$\epsilon(N) = \frac{d_\delta}{\sqrt{N}} + \frac{f_\delta}{N},$$

where

$$d_\delta = \sqrt{2S_N \ln 3/\delta}, \qquad f = 3\mathbf{R} \ln 3/\delta,$$

No reasonably simple closed-form solution is available in this case, requiring a linear search of the batch size N^* maximizing Υ_δ.

The cost function to optimize becomes (already optimized w.r.t the step size):

$$\Upsilon_\delta(\Lambda^*, N) = \frac{\left(\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty - \sqrt{\frac{2S_N \ln 3/\delta}{N}} - \frac{3\mathbf{R} \ln 3/\delta}{N}\right)^4}{4cN\left(\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty + \sqrt{\frac{2S_N \ln 3/\delta}{N}} + \frac{3\mathbf{R} \ln 3/\delta}{N}\right)^2}.$$

First of all, Assumption 4.2.7 gives the following constraint:

$$N \geqslant N_0 := \left(\frac{d_\delta + \sqrt{d_\delta^2 + 4f_\delta \left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty}}{2\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty}\right)^2,$$

so N_0 can be used as the starting point of the linear search. Computing the derivative w.r.t N we obtain two distinct factors for the numerator:

$$\left(d_\delta \sqrt{N} + f_\delta - \left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty N\right)^3,$$

$$\left(2d_\delta^2 N + 5d_\delta f_\delta \sqrt{N} + 3d_\delta \left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty N^{3/2} + 3f_\delta^2 + 6f \left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty N - \left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty^2 N^2\right).$$

The first one gives again N_0, which is the global minimum. By applying Descartes' rule of signs to the other one, seen as a polynomial in $\sqrt{N}$, we see that it gives just one root. The exact expression of this maximum is too big to be reported and too long to code, but its uniqueness, together with the fact that Υ_δ goes to 0 as N goes to infinity, justifies the following methodology: to find N^*, start from N_0 and stop as soon as the value of the cost function $\Upsilon(\Lambda^*, N)$ begins to decrease.

Note also that the optimal step size is no longer constant: it can be computed with the expression given in Corollary 4.2.8 by setting $\epsilon := \epsilon(N^*)$. Finally, as for the Hoeffding's bound, the range $\mathbf{R}$ can be upper bounded exactly or estimated from samples.

5 NUMERICAL SIMULATIONS

In this chapter we describe some experiments performed to test our safe methods. We compare the different variants of our algorithm to observe their performance in practice. We also compare our method with older or more common approaches to see how much the theoretical improvements obtained in the previous chapter translate into better practical results.

In Section 5.1 we use a simple simulated control task, one-dimensional Linear-Quadratic Gaussian regulation (LQG), to compare the different variants of our algorithm. In Section 5.2 we use a two-dimensional variant of the same problem to show the advantages of our method over previous approaches. In Section 5.3 we use a continuous action variant of the popular cart-pole task to improve a baseline policy after a change has been introduced to the environment, to show the practical advantages of our method.

5.1 ONE-DIMENSIONAL LQG

In this section we test the proposed methods on the one-dimensional LQG problem [17]. We use this task as a testing ground because it is simple, all the constants involved in our bounds can be computed exactly, and the optimal parameter θ^* is available as a reference.

The LQG problem is a continuous MDP defined by transition model:

$$s_t \sim \mathcal{N}(s_t + a_t, \sigma_0^2),$$

reward function:

$$r_t = -0.5(s_t^2 + a_t^2),$$

and Gaussian policy:

$$a_t \sim \mathcal{N}(\theta \cdot s_t, \sigma^2).$$

Intuitively, the problem is to bring to zero a system's state, in the presence of noise, with a cost for acting on the system. The policy has a single parameter θ. The optimal value θ^* can be computed exactly, but the problem can be used as a benchmark for RL algorithms.

In all our simulations we use $\sigma_0 = 0$, since all the noise can be modeled on the agent's side without loss of generality. Both action and state variables are bounded to the interval $[-2, 2]$ and the initial state is drawn uniformly at random. We use a discount factor $\gamma = 0.9$, which gives as optimal parameter $\theta^* \approx -0.59$, corresponding to expected performance $J(\theta^*) \approx -13.21$. The expected performance is

very pessimistic, since it does not take into account the bound on actions and states and the limited length of the task horizon, but equally captures the progress of the real performance. A monotonically improving expected performance is a guarantee that the policy is improving without oscillation, even if the measured performance is noisy due to stochasticity in the environment and in the policy itself.

In our experiments, we are interested both in the convergence speed and in policy oscillation. We call *improvement ratio* the ratio of policy updates that does not result in a worsening of the expected performance. First of all, we want to analyze how the choice of fixed step sizes and batch sizes may affect the improvement ratio and how much it depends on the variability of the trajectories (that in this case is due to the variance of the policy σ). Table 5.1 shows the improvement ratio for two parameterizations ($\sigma = 0.5$ and $\sigma = 1$) when various constant step sizes and batch sizes are used, starting from $\theta = -0.55$ and stopping after a total of one million trajectories.

Table 5.1: Improvement ratio of the policy updates for different policy standard deviation σ, fixed batch size N and fixed step size α, using the G(PO)MDP gradient estimator.

		$\sigma = 0.5$			$\sigma = 1$		
		N = 10000	N = 1000	N = 100	N = 10000	N = 1000	N = 100
α	1e-3	95.96%	52.85%	49.79%	24.24%	37.4%	50.4%
	1e-4	100%	73.27%	51.41%	100%	27.03%	46.08%
	1e-5	98.99%	81.88%	55.69%	100%	99.9%	39.04%
	1e-6	100%	83.88%	58.44%	100%	100%	86.04%

As expected, small batch sizes combined with large step sizes lead to low improvement ratios. However, the effect of the meta-parameters is non-trivial and problem-dependent, justifying the need for an adaptive method.

We now proceed to test the methods described in Chapter 4. In all the following simulations we use $\sigma = 1$ and start from $\theta = 0$, stopping after a total of 30 million trajectories.

We start by testing Algorithm 4.1 with Chebyshev's bound as proposed in Section 4.3.1. To estimate the gradient, we use the REINFORCE and G(PO)MDP gradient estimators. In both cases, we use optimal baseline from [17]. Figure 5.1 shows the expected performance over sample trajectories for the two gradient estimators and for different values of δ, the probability with which worsening updates are allowed to take place.

To make this and the following figures more interpretable, some words must be spent on how data were processed. Each learning iteration corresponds to a single value of θ, with a corresponding expected

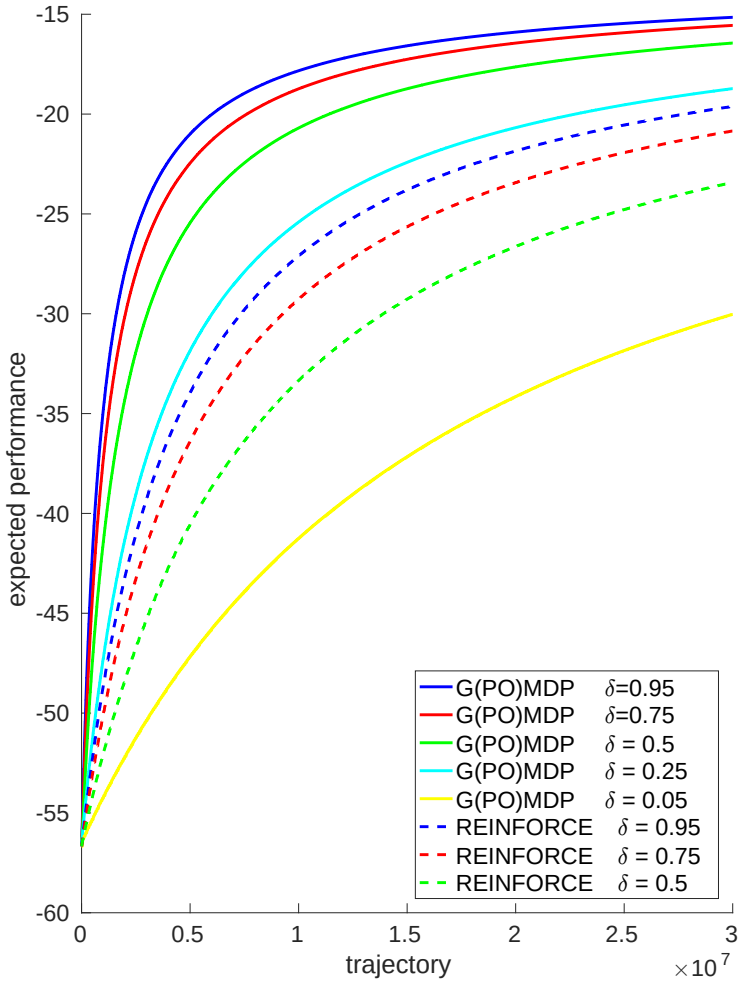


Figure 5.1: Expected performance over sample trajectories in the one-dimensional LQG task, using G(PO)MDP and REINFORCE (dashed) gradient estimators and Chebyshev bound, for different values of δ. Expected performance is computed for each parameter update. Data are then scaled to account for the different batch sizes. All results are averaged over 5 runs of 30 million total trajectories each.

performance. The latter was scaled over time, simply by maintaining the value for all the trajectories of the batch (a closer look to the plot would reveal a series of steps of variable length and height). The horizontal scaling was performed to better capture the improvement of performance over time.

First of all, we can see that in all cases the performance curves are very regular. This is due to the small step sizes selected by the algorithm, which are of the order of 10^{-5}.

Table 5.2: Average performance in the one-dimensional LQG task for different gradient estimators, statistical bounds and values of δ. All results are averaged over 5 runs (95% confidence intervals are reported). The abbreviation 'e.r.' stands for 'empirical range'

Estimator	Bound	δ	$\overline{\Upsilon}$	Confidence interval
REINFORCE	Chebyshev	0.95	-11.3266	[-11.3277; -11.3256]
REINFORCE	Chebyshev	0.75	-11.4303	[-11.4308; -11.4297]
REINFORCE	Chebyshev	0.5	-11.5947	[-11.5958; -11.5937]
G(PO)MDP	Chebyshev	0.95	-10.6085	[-10.6087; -10.6083]
G(PO)MDP	Chebyshev	0.75	-10.7141	[-10.7145; -10.7136]
G(PO)MDP	Chebyshev	0.5	-10.9036	[-10.9040; -10.9031]
G(PO)MDP	Chebyshev	0.25	-11.2355	[-11.2363; -11.2346]
G(PO)MDP	Chebyshev	0.05	-11.836	[-11.8368; -11.8352]
G(PO)MDP	Hoeffding	0.95	-11.914	[-11.9143; -11.9136]
G(PO)MDP	Bernstein	0.95	-10.2159	[-10.2162; -10.2155]
G(PO)MDP	Hoeffding (e. r.)	0.95	-9.8582	[- 9.8589; - 9.8574]
G(PO)MDP	Bernstein (e. r.)	0.95	-9.6623	[- 9.6619; - 9.6627]

In general REINFORCE performs worse than G(PO)MDP due to its larger variance. Larger values of δ lead to better performance. Notice that an improvement ratio of 1 is achieved also with large values of δ. This is due to the fact that the bounds used in the development of our method are not tight. Being the method this conservative, in practical applications δ can be set to a high value to improve the convergence rate.

The next step is to compare the Chebyshev's bound approach to the other methods proposed in Section 4.3. We stick to the configuration that performed the best in the previous experiment: G(PO)MDP to estimate the gradient and $\delta = 0.95$. Figure 5.2 compares the performance of the different concentration bounds.

As expected, Bernstein's bound performs better than Chebyshev's, especially in the empirical range version. The rigorous version of Hoeffding's bound performs very poorly, while the one using the empirical range is almost as good as the corresponding Bernstein method. This is due to the fact that the bound on the gradient estimate range is very loose, since it accounts also for unrealistic combinations of state, action and reward.

To better capture the performance of the different variants of the algorithm in a real-time scenario, we define a metric $\overline{\Upsilon}$, that is obtained by averaging the real performance (measured during learning) over

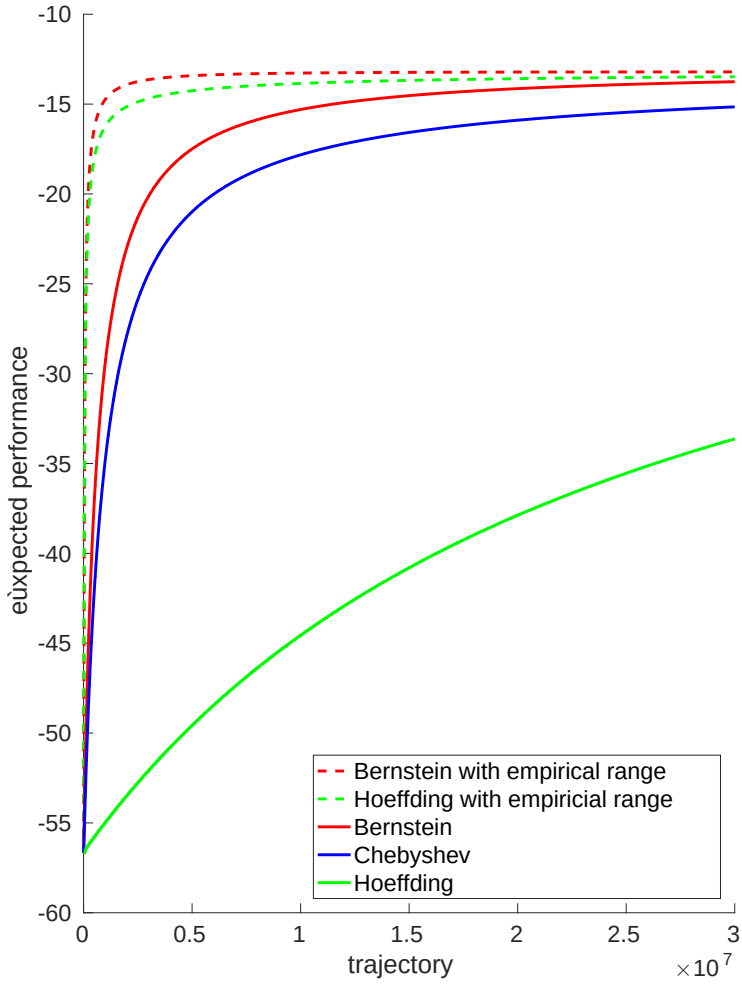


Figure 5.2: Comparison of the performance of different statistical bounds in the one-dimensional LQG task, using the G(PO)MDP gradient estimator and $\delta = 0.95$. All results are averaged over 5 runs.

all the trajectories, coherently with the cost function used to derive the optimal batch size. The results are reported in Table 5.2.

Finally, it is interesting to look at the trend of the batch size over learning iterations. Figures 5.3 and 5.4 show the trend of the batch size for the different concentration bounds.

As expected, the batch size increases as the optimal parameter is approached, since positive improvements require more and more accuracy. The trend is evidently superlinear, with the exception of Hoeffding's bound with the theoretical range bound, which however selects very high batch sizes from the beginning.

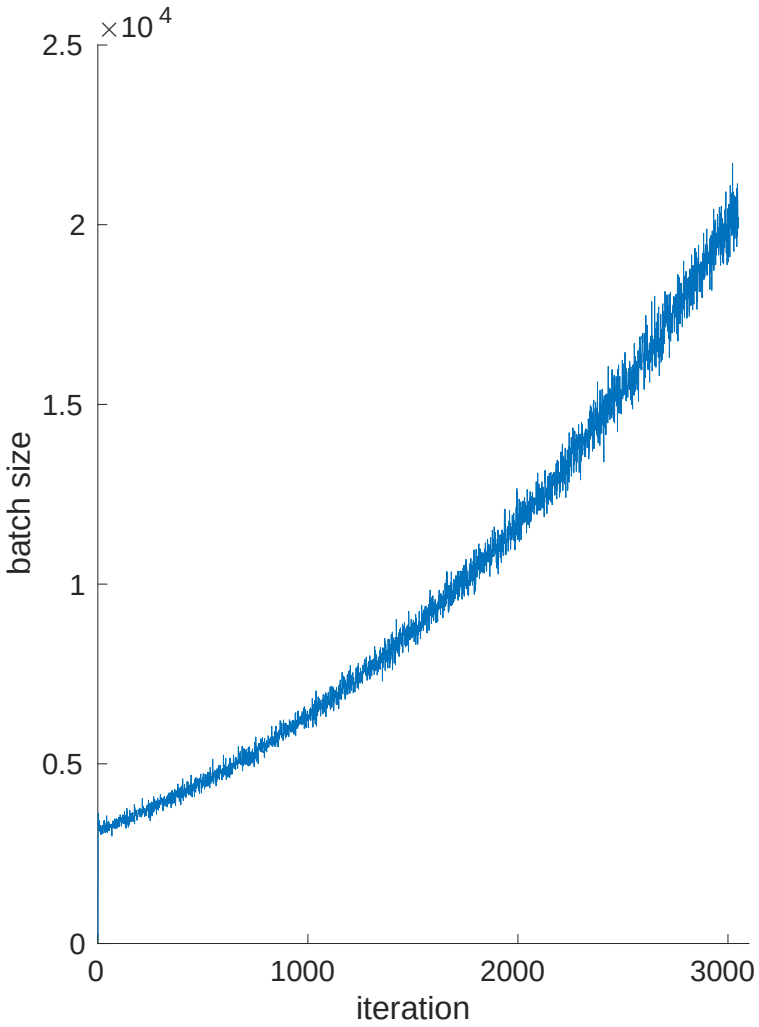
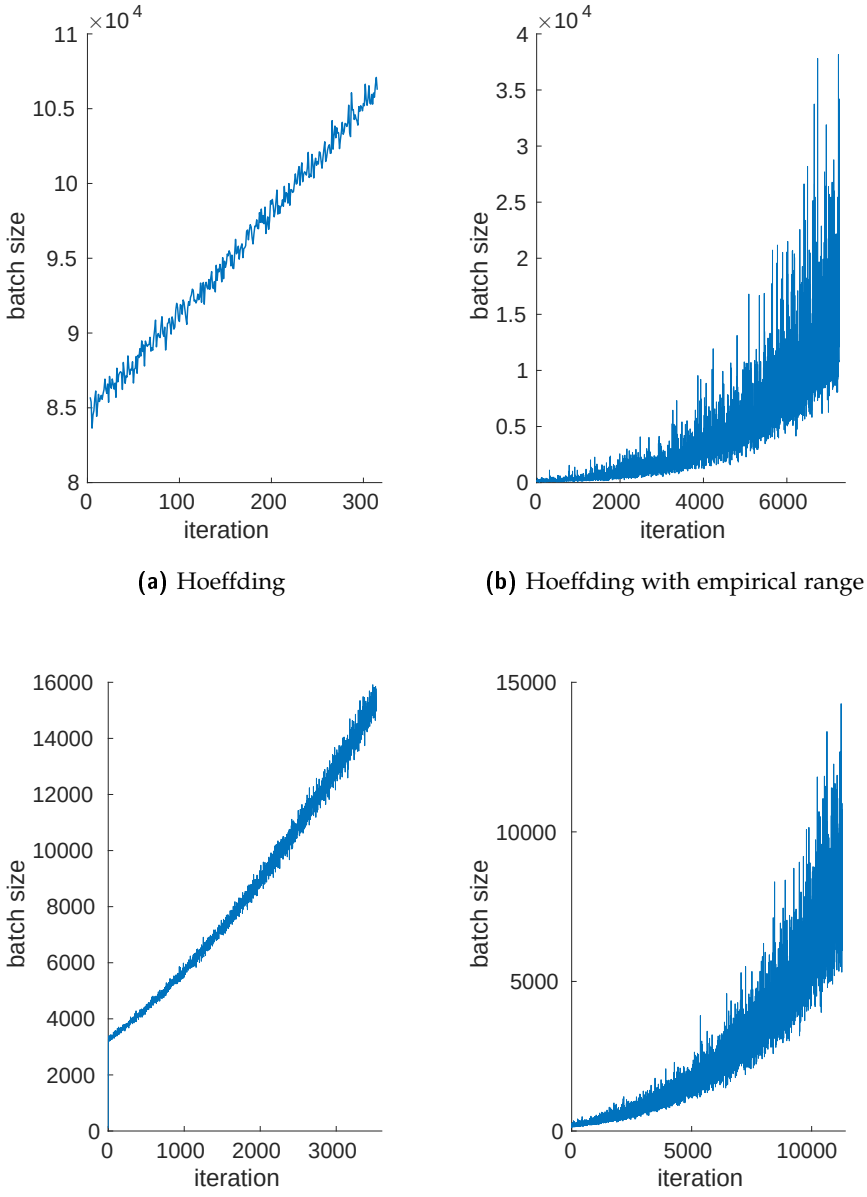


Figure 5.3: Adaptive batch size over learning iterations in the one-dimensional LQG task, using the G(PO)MDP gradient estimator, Chebyshev's bound and $\delta = 0.95$.



(a) Hoeffding

(b) Hoeffding with empirical range

(c) Bernstein

(d) Bernstein with empirical range

Figure 5.4: Adaptive batch size over learning iterations in the one-dimensional LQG task for different concentration bounds. In all the cases, the G(PO)MDP gradient estimator and $\delta = 0.95$ were used.

In all the experiments, the step size α is of the order of 10^{-5} and does not change significantly over learning iterations.

5.2 MULTI-DIMENSIONAL LQG

The one-dimensional LQG problem was a good benchmark to compare the performance of different concentration bounds, but does not capture at all the coordinate descent aspect of our algorithm. Luckily, LQG can easily be generalized to any number of dimensions m.

Following the approach of [23], we define multi-dimensional LQG as an MDP with transition model:

$$\mathbf{s}_{t+1} \sim \mathcal{N}(\mathbf{s}_t + \mathbf{a}_t, C_0),$$

reward function:

$$r_t = -0.5(\mathbf{s}_t^T Q \mathbf{s}_t + \mathbf{a}_t^T R \mathbf{a}_t),$$

and Gaussian policy:

$$\mathbf{a}_t \sim \mathcal{N}(\theta \cdot \mathbf{s}_t, C),$$

where both $\mathbf{s}$ and $\mathbf{a}$ are vectors of size m, C_0 and C are covariance matrices, Q is a semidefinite matrix and R is a symmetric positive definite matrix. All the matrices are $m \times m$. Note that also θ is an $m \times m$ matrix:

$$\theta = \begin{bmatrix} \theta_{11} & \theta_{12} \\ \theta_{21} & \theta_{22} \end{bmatrix}.$$

Similarly to the one-dimensional case, we set $C_0 = 0$ without loss of generality. Matrices Q and R can be used to define different objectives.

5.2.1 *Independent actions in two dimensions*

By setting Q and R to the identity matrix and using a diagonal covariance C, the result are two simultaneous, independent one-dimensional LQG problems. Although this fact is intuitive, it may not be trivial for a learning algorithm to exploit it. In particular, we consider the two dimensional case ($m = 2$), with covariance matrix:

$$C = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}.$$

In particular, in our experiments, we use $\sigma_1 = \sigma_2 = 1$ and $\gamma = 0.95$. With these parameters, the optimal parameter matrix is (approximately):

$$\theta^* \simeq \begin{bmatrix} -0.588 & 0 \\ 0 & -0.588 \end{bmatrix},$$

with corresponding expected performance $J_\mu(\theta^*) \simeq -422.616$. The fact that θ is diagonal is due precisely to the independence of the two one-dimensional LQG tasks.

We first use this task to provide empirical evidence to the claim of Corollary 4.2.5, i.e., to show that the adaptive step size that we proposed is an improvement over the existing result that we generalized. We run two simulations, one with the scalar adaptive step size $\hat{\alpha}^*$ proposed in [24], which we report below, and one with the non-scalar adaptive step size Λ^* of Corollary 4.2.8:

$$\hat{\alpha}^* = \frac{(1-\gamma)^3\sqrt{2\pi}\sigma^3 \left\|\underline{\hat{\nabla}_\theta J_\mu}(\theta)\right\|_2^2}{(\gamma\sqrt{2\pi}\sigma) + 2(1-\gamma)|\mathcal{A}|)RM_\phi^2 \left\|\overline{\hat{\nabla}_\theta J_\mu}(\theta)\right\|_1^2}.$$

In both the simulations we use the G(PO)MDP gradient estimator with variance-minimizing baseline, Chebyshev's bound, $\delta = 0.95$ and fixed batch size $N = 2000$. We start from $\theta = \underline{0}$ and stop after a total of 30 million trajectories. The selected batch size happens to satisfy Assumption 4.2.7 for the duration of the experiment. This is necessary to preserve the performance improvement guarantee in both settings. Figure 5.5 shows the expected performance over trajectories for the two experiments.

In both cases, an improvement ratio of 100% was achieved. Results on performance show that, indeed, the non-scalar step size outperforms its scalar counterpart. To get a better insight on how this happens, it is useful to compare the trend of the adaptive step size in the two cases. Figure 5.6 compares the trend of $\hat{\alpha}^*$ and $\|\Lambda^*\|_\infty$.

We can see that, although it decreases faster, the non-scalar step size dominates (in magnitude) the scalar one. Basically, the algorithm from [24] needs to be more conservative, since more parameters are updated at each learning iteration and their penalties on the guaranteed improvement sum up.

Another interesting difference between the two methods is in the shape of the learned policy parameter θ. In any stochastic policy gradient methods, the gradient components relative to θ_{12} and θ_{21} are small if compared to the other two, but not zero. This is due to noise and the result is that, even by starting from $\theta = \underline{0}$, the learned parameter will not be a diagonal matrix like the optimal one. On the contrary, a coordinate descent method like ours will never choose to update θ_{12} or θ_{21}, because the gradient components associated to θ_{11} and θ_{22} are much larger. Starting from $\theta = \underline{0}$, a diagonal matrix is obtained. Although this does not affect performance significantly, it shows how our algorithm is able to completely avoid useless changes in the current policy, which are potential causes of oscillation. In particular, the final parameter obtained with the scalar step size $\hat{\alpha}^*$ was:

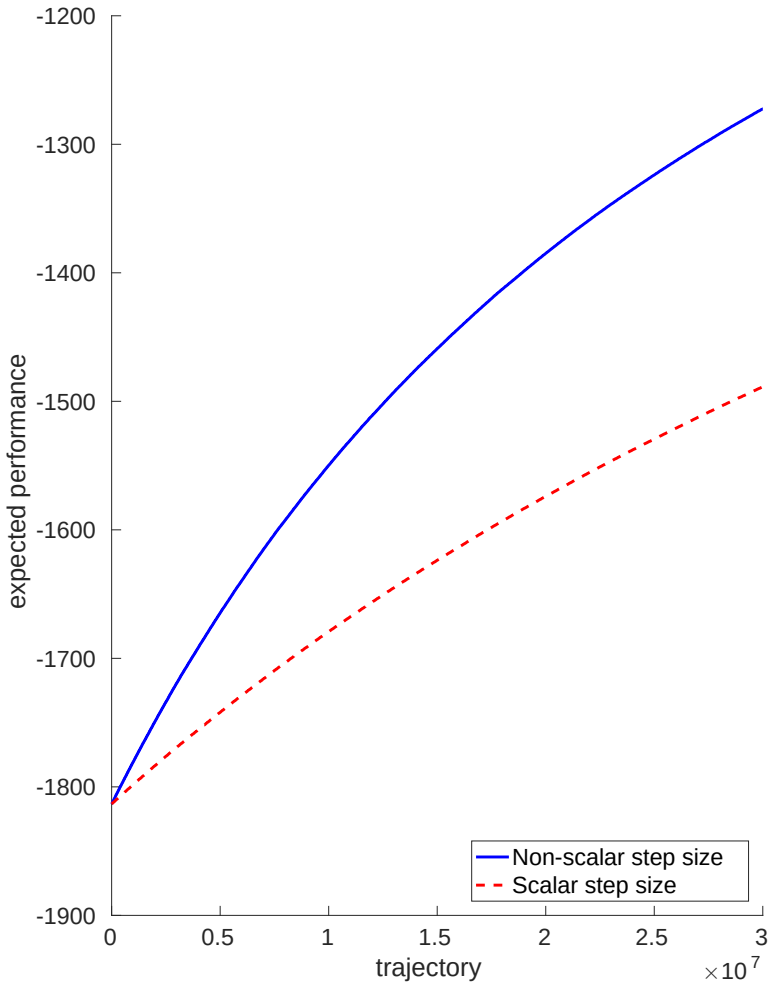


Figure 5.5: Expected performance over sample trajectories in the two-dimensional LQG task, using G(PO)MDP, Chebyshev bound, $\delta = 0.95$ and fixed $N = 2000$. The performance of the adaptive non-scalar step size proposed in this work is compared with the scalar step size from [24], of which the former is a generalization.

$$\theta = \begin{bmatrix} -0.014676 & 1.4e-06 \\ 1.7e-06 & -0.013678 \end{bmatrix},$$

while, with our non-scalar step size Λ^*, we obtained:

$$\theta = \begin{bmatrix} -0.0275 & 0 \\ 0 & -0.0274 \end{bmatrix},$$

which is indeed diagonal.

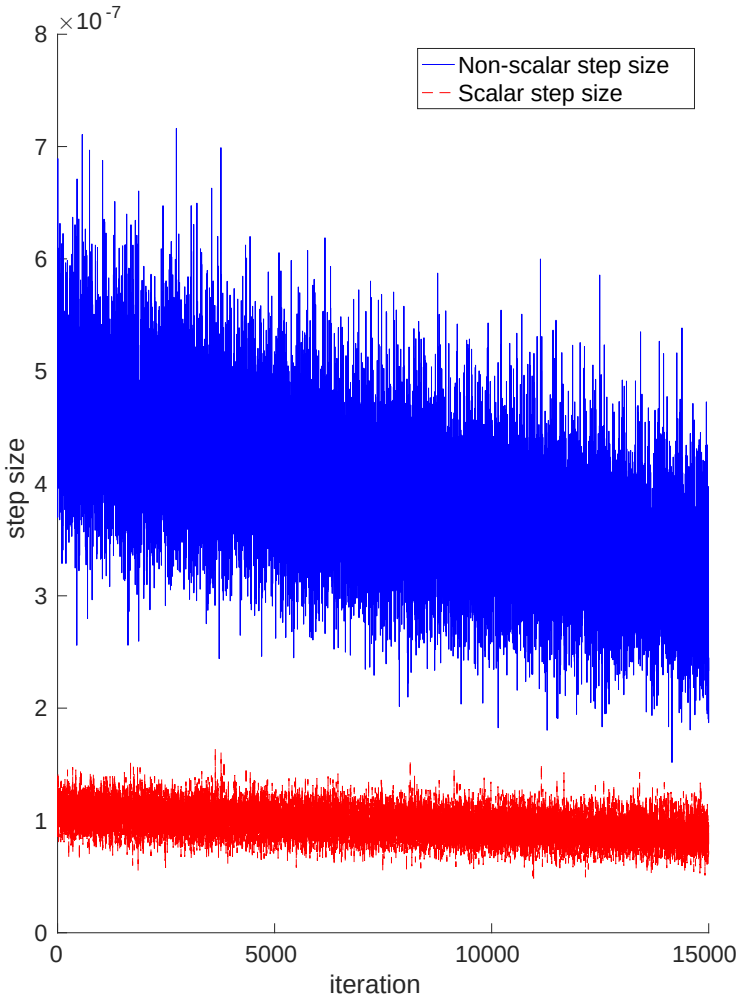


Figure 5.6: Magnitude of the step size over learning iterations. The adaptive non-scalar step size proposed in this work is compared with the adaptive scalar step size from [24], of which the former is a generalization.

We then test our adaptive batch-size on the same task. We use the G(PO)MDP gradient estimator with optimal baseline, Bernstein's bound with empirical range and $\delta = 0.95$. Figure 5.7 shows the evolution of the batch size over the learning process in the adaptive case.

The adaptive batch size goes from $N \simeq 500$ in the first learning iterations, to $N \simeq 2500$ in the end. The step size takes values of the order of $1e-6$ for the whole process. We compare the performance of our algorithm with the one of vanilla policy gradient with fixed step size and batch size. Since we want to focus our analysis to the effects of the batch size, we use G(PO)MDP with optimal baseline

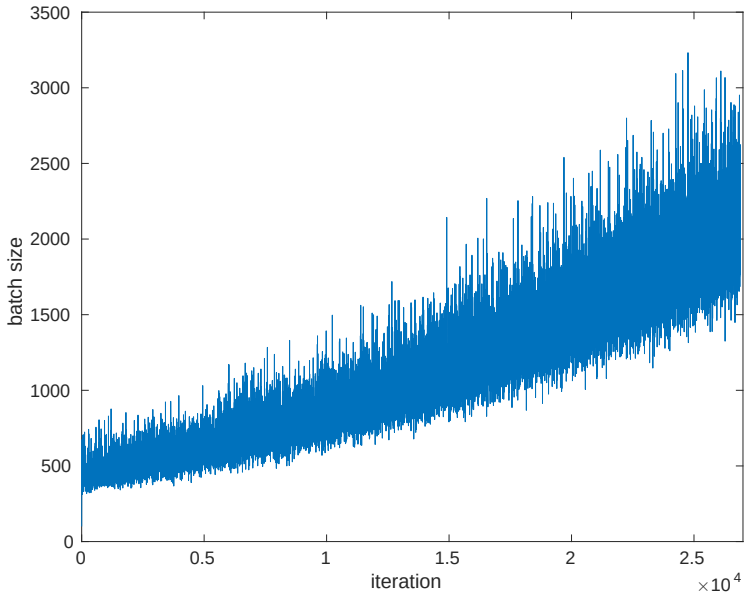


Figure 5.7: Adaptive batch size over learning iterations for the two-dimensional LQG task. G(PO)MDP gradient estimator with optimal baseline, Bernstein's bound with empirical range and $\delta = 0.95$ were used.

and $\alpha = 1e-6$. We run two simulations, one with $N = 1000$ and one with $N = 100$. Figure 5.8 compares expected performance over learning iterations, while table 5.3 reports average performance $\overline{\Upsilon}$ and improvement ratio for the three cases.

Table 5.3: Average performance and improvement ratio for different simulations on the two-dimensional LQG task, using G(PO)MDP. The adaptive batch size is computed using Bernstein's bound with empirical range and $\delta = 0.95$. The fixed batch sizes are used in conjunction with $\alpha = 1e-6$.

N	$\overline{\Upsilon}$	Improvement Ratio
Adaptive	-20.50	100%
1000	-22.39	100%
100	-19.77	89%

Results show how a large, fixed batch size can badly affect the convergence rate. Our algorithm is able to outperform the $N = 1000$ case by safely employing smaller batch sizes at the beginning of the learning process, turning to larger ones only when it is necessary. For this task, a small batch size, such as $N = 100$, yields a better average performance than our algorithm, but it introduces oscillation, witnessed

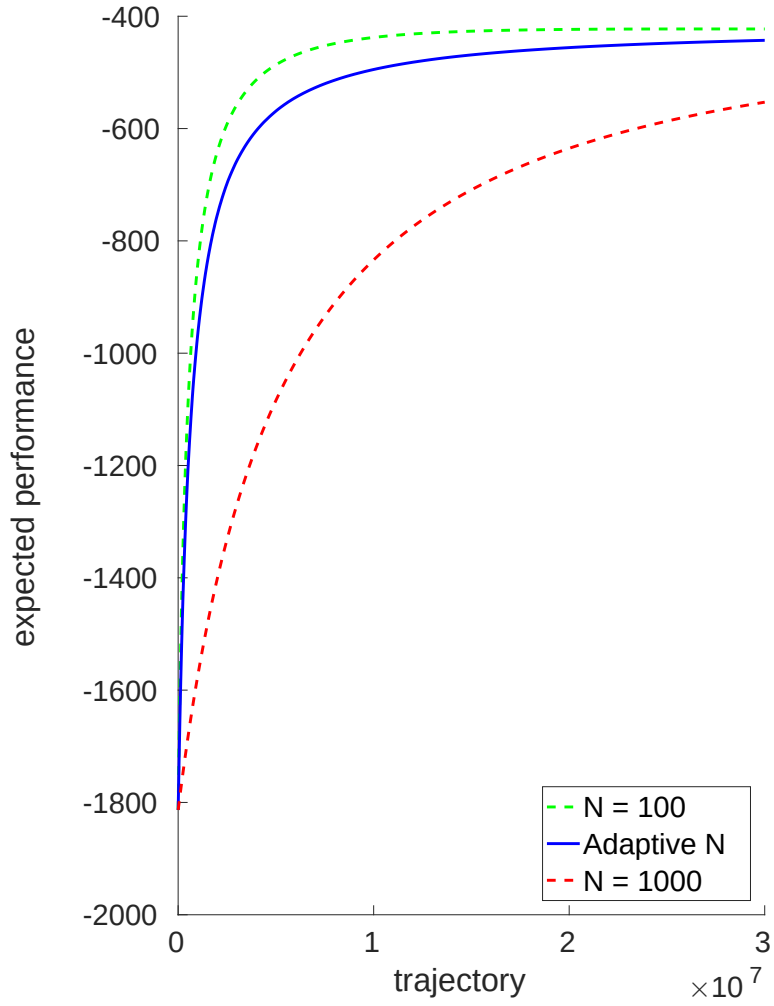


Figure 5.8: Expected performance over sample trajectories in the two-dimensional LQG task. The adaptive algorithm with Bernstein's bound and $\delta = 0.95$ is compared with vanilla policy gradient with fixed $N = 100, 1000$ and $\alpha = 1e-6$.

by a less-than-one improvement ratio. These results show how an adaptive batch size, such as the one employed by our algorithm, is able to guarantee monotonic improvement without needlessly sacrificing convergence speed.

5.3 CART-POLE

In this section we test our algorithm on a more challenging simulated control task to give an idea of how it could be useful in practice. We use our algorithm to improve an existing policy after a change has

been introduced in the environment, to highlight the advantages of our safe approach.

The cart-pole task is a popular RL benchmark problem, of which we use a continuous action variant. An inverted pendulum, the pole, is attached to a cart. A force can be applied to the cart, in order to maintain the pole in a vertical position as long as possible. If the pole becomes too much inclined or the cart goes too far to the left or to the right, the task ends.

In our setting, the cart has a mass of 1Kg, the pole is 1m long and has a mass of 0.1Kg. The agent's state is the tuple:

$$\mathbf{s}_t = [x, \dot{x}, \beta, \dot{\beta}],$$

where x is the position of the cart, $\dot{x}$ is the velocity of the cart, β is the angle that the pole forms with the vertical and $\dot{\beta}$ is the angular velocity of the pole. The state is initialized at random in a small neighborhood of $\underline{0}$. The action is scalar:

$$a_t \in [-10, 10],$$

and represents the magnitude of the force applied horizontally to the cart, in newtons. A constant reward of 1 is obtained at each time step, such as longer episodes yield better performance. The trajectory has a maximum length of $H = 200$ time steps, but it ends as soon as $\beta \notin [-41.8°, 41.8°]$ or $x \notin [-2.4m, 2.4m]$.

We use a Gaussian policy:

$$a_t \sim \mathcal{N}(\theta^T \mathbf{s}_t, \sigma),$$

with $\sigma = 1$ and starting from $\theta = \underline{0}$.

We first learn a good policy with a vanilla policy gradient algorithm, using the G(PO)MDP gradient estimator with variance-minimizing baseline, fixed step size $\alpha = 0.01$ and batch size $N = 4000$. We stop once an average performance of $J_\mu(\theta) = 195$ over the batch is achieved. This result is obtained in 72 iterations. The obtained policy will be referred to as baseline policy.

At this point we introduce a change in the environment, by doubling the length of the pole, keeping its mass constant. If the baseline policy is used in this new setting, the performance immediately drops. Instead, we try to adapt to the change by learning a new policy. This is representative of the scenario presented in Chapter 1, where an available policy must be improved to adapt to small changes in the environment.

We run two simulations: the first with the same algorithm used to learn the baseline policy, the second with our adaptive algorithm, using Bernstein's bound with empirical range and $\delta = 0.95$. Figure 5.9 shows the performance over trajectories of the two approaches (average performance over the batch is used as representative for all the

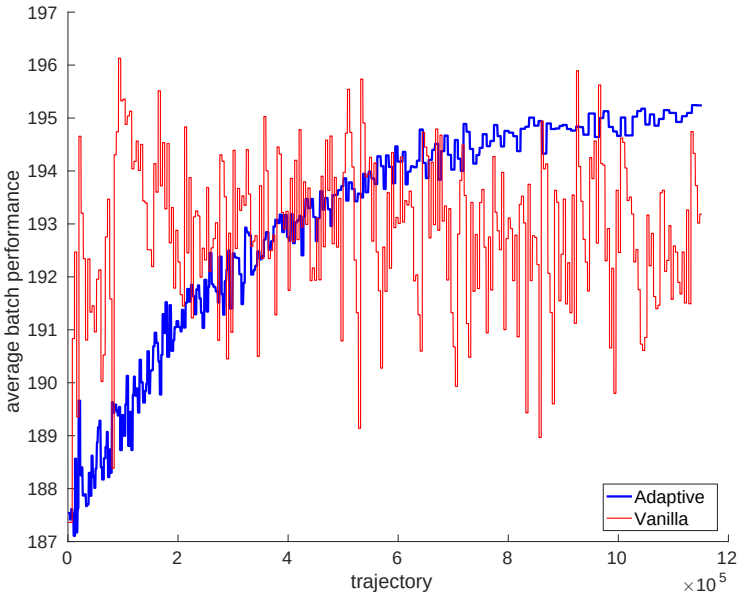


Figure 5.9: Expected performance over sample trajectories in the modified cart-pole task. The adaptive algorithm with Bernstein's bound and $\delta = 0.95$ is compared with vanilla policy gradient with fixed $N = 4000$ and $\alpha = 1e-2$.

trajectories in the batch), while Table 5.4 reports average performance and improvement ratio for the two cases. Note that, since the exact value of the expected performance is not available for this task, we are not able to isolate true performance oscillation from noise due to stochasticity in the environment and in the policy itself. However, the improvement ratio computed on the average performance is still indicative of how steadily the policy is improving.

Table 5.4: Average performance and improvement ratio for different simulations in the modified cart-pole task, using G(PO)MDP. The adaptive batch size is computed using Bernstein's bound with empirical range and $\delta = 0.95$. The fixed batch size is used in conjunction with $\alpha = 1e-2$.

N	$\overline{\Upsilon}$	Improvement ratio
Adaptive	193.12	53.01%
4000	192.86	44.76%

Results show how the non-safe approach, although reaching a good performance very quickly, is subject to heavy oscillations, which affect the long term performance. To get the best from this algorithm, a human expert should stop the learning process at the right moment.

Instead, our safe algorithm, although taking much longer to reach a good performance, is subject to less oscillations and is able to maintain it indefinitely. This allows for a more autonomous agent, that can react to changes in the environment without any human intervention.

Once again, examining the learned parameters leads to interesting insights: our algorithm never updates the components of θ corresponding to β and $\dot{\beta}$, evidently because they are not relevant in adjusting to the particular change that was introduced to the environment. As already observed in Section 5.2, the coordinate descent approach is able to completely avoid some unnecessary sources of oscillation.

Although definitely artificial, the cart-pole example shows the possible advantages of employing a safe policy gradient method to a realistic problem, and highlights the main strengths of our proposed algorithm.

6 CONCLUSIONS

In this thesis we were able to devise an adaptive policy gradient algorithm for Gaussian policies that guarantees monotonic performance improvement with high probability. This algorithm performs coordinate descent and automatically selects both the step size for the parameter updates and the batch size used to collect samples.

Experiments showed that the algorithm is effective in guaranteeing monotonic improvement, but very conservative in the selection of the batch size. More empirical variants were proposed to address this problem, with satisfying results. However, there is evidence that the method is still quite conservative. First of all, the worsening probability δ turned out to have no actual effect on the improvement ratio. Even when values of δ close to one are selected, the algorithm is conservative enough to avoid oscillation at all. This deprives us of a degree of freedom that could be used to regulate the behavior of the algorithm depending, for instance, on risk aversion or on the degree of non-stationarity of the environment. Experiments also show how a manually selected batch size can still outperform our method in terms of average performance in many cases. Although small batch sizes do not guarantee monotonic improvement, not always the entity of oscillation has a relevant impact on performance. If we had more control on the wariness of the algorithm, in some applications it would be useful to allow some worsening updates, in order to speed up convergence without compromising long term performance.

Future work could focus on improving the method by devising tighter bounds on the performance improvement. This would allow to guarantee safety with smaller batches, improving convergence speed, and to gain more control on the degree of safety that needs to be imposed.

The bounds proposed in Section 4.3.1 for the REINFORCE and the G(PO)MDP gradient estimators could be improved by taking into consideration also the effect of variance-minimizing baselines. Additionally, similar ad-hoc bounds could be devised for the other gradient estimation algorithms described in Section 2.3.

Another possibility for future work is to extend the results to other classes of policies. The current approach can be applied only to Gaussian policies with linear expectation and constant standard deviation. Although this kind of policy is broadly used in control tasks, it does not cover all the possible applications of policy gradient methods. One idea is to generalize the present results, from more complex Gaussian policies to generic parametrized policies. It would also be

useful to adapt the results to other specific classes of policies that are commonly used in practice. For instance, the Softmax policy is a common choice for tasks characterized by a discrete action space. Appendix B shows a first attempt in this direction, highlighting the main difficulties that arise when the approach used for Gaussian policies has to be adapted to the new class.

Other, more ambitious developments may include the analysis of other parameters and their relationship with the ones that have already been considered. In particular, exploration represents a major problem for policy gradient, as pointed out in [13]. Including exploration-controlling parameters, such as the standard deviation of the Gaussian policy, in the analysis may lead to faster safe policy gradient algorithms.

BIBLIOGRAPHY

[1] Yasin Abbasi-Yadkori, Peter L Bartlett, and Stephen J Wright. "A Fast and Reliable Policy Improvement Algorithm." In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. 2016, pp. 1338–1346 (cit. on p. 22).

[2] Shun-Ichi Amari. "Natural gradient works efficiently in learning." In: *Neural Computation* 10.2 (Feb. 1998), pp. 251–276. ISSN: 0899-7667 (cit. on p. 16).

[3] J. Baxter and P. L. Bartlett. "Infinite-Horizon Policy-Gradient Estimation." In: *Journal of Artificial Intelligence Research* 15 (2001), pp. 319–350 (cit. on p. 65).

[4] Jonathan Baxter and Peter L. Bartlett. "Infinite-Horizon Policy-Gradient Estimation." In: *Journal of Artificial Intelligence Research* 15 (2001), pp. 319–350 (cit. on p. 16).

[5] DimitriP. Bertsekas. "Approximate policy iteration: a survey and some new methods." English. In: *Journal of Control Theory and Applications* 9.3 (2011), pp. 310–335. ISSN: 1672-6340 (cit. on p. 19).

[6] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004. ISBN: 0521833787 (cit. on p. 30).

[7] Peter Dayan. "The convergence of TD(λ) for general λ." In: *Machine Learning* 8.3 (1992), pp. 341–362. ISSN: 1573-0565 (cit. on p. 10).

[8] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. "A Survey on Policy Search for Robotics." In: *Foundations and Trends in Robotics* 2.1-2 (2013), pp. 1–142 (cit. on pp. 1, 11).

[9] Peter W. Glynn. "Likelihood Ratio Gradient Estimation for Stochastic Systems." In: *Commun. ACM* 33.10 (Oct. 1990), pp. 75–84. ISSN: 0001-0782 (cit. on p. 13).

[10] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients." In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 42.6 (2012), pp. 1291–1307 (cit. on p. 17).

[11] Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. "Convergence of stochastic iterative dynamic programming algorithms." In: *Advances in neural information processing systems*. 1994, pp. 703–710 (cit. on p. 10).

[12] Sham Kakade. "A Natural Policy Gradient." In: *Advances in Neural Information Processing Systems 14 (NIPS 2001)*. Ed. by Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani. MIT Press, 2001, pp. 1531–1538 (cit. on p. 16).

[13] Sham Kakade and John Langford. "Approximately Optimal Approximate Reinforcement Learning." In: *ICML*. Morgan Kaufmann, 2002, pp. 267–274 (cit. on pp. 2, 21, 23, 30, 56).

[14] Jens Kober and Jan Peters. "Policy Search for Motor Primitives in Robotics." In: *Advances in Neural Information Processing Systems 21*. Vol. 21. 2008, pp. 849–856 (cit. on p. 12).

[15] Volodymyr Mnih, Csaba Szepesvári, and Jean-Yves Audibert. "Empirical Bernstein Stopping." In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: ACM, 2008, pp. 672–679. ISBN: 978-1-60558-205-4 (cit. on p. 37).

[16] Jorge J. Moré and David J. Thuente. "Line Search Algorithms with Guaranteed Sufficient Decrease." In: *ACM Trans. Math. Softw.* 20.3 (Sept. 1994), pp. 286–307. ISSN: 0098-3500 (cit. on pp. 11, 23).

[17] J. Peters and S. Schaal. "Reinforcement Learning of Motor Skills with Policy Gradients." In: *Neural Networks* 21.4 (May 2008), pp. 682–697 (cit. on pp. 39, 40).

[18] Jan Peters, Katharina Mülling, and Yasemin Altun. "Relative Entropy Policy Search." In: *AAAI*. AAAI Press, 2010 (cit. on p. 12).

[19] Jan Peters and Stefan Schaal. "Natural Actor-Critic." In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190 (cit. on pp. 12, 16).

[20] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients." In: *Neural Networks* 21.4 (2008), pp. 682–697 (cit. on pp. 15, 16).

[21] Marek Petrik, Mohammad Ghavamzadeh, and Yinlam Chow. "Safe policy improvement by minimizing robust baseline regret." In: *Advances in Neural Information Processing Systems* (2016) (cit. on p. 26).

[22] M. S. Pinsker. *Information and Information Stability of Random Variables and Processes*. Moskva: Izv. Akad. Nauk, 1960 (cit. on p. 62).

[23] Matteo Pirotta, Simone Parisi, and Marcello Restelli. "Multi-objective Reinforcement Learning with Continuous Pareto Frontier Approximation." In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 2928–2934. ISBN: 0-262-51129-0 (cit. on p. 46).

[24] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. "Adaptive Step-Size for Policy Gradient Methods." In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Curran Associates, Inc., 2013, pp. 1394–1402 (cit. on pp. 23, 24, 27–30, 36, 47–49, 61, 62, 68).

[25] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. "Policy gradient in Lipschitz Markov Decision Processes." In: *Machine Learning* 100.2-3 (2015), pp. 255–283 (cit. on pp. 2, 24).

[26] Matteo Pirotta, Marcello Restelli, Alessio Pecorino, and Daniele Calandriello. "Safe Policy Iteration." In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. 3. JMLR Workshop and Conference Proceedings, 2013, pp. 307–315 (cit. on p. 22).

[27] Martin L. Puterman and Shelby L. Brumelle. "On the Convergence of Policy Iteration in Stationary Dynamic Programming." In: *Math. Oper. Res.* 4.1 (Feb. 1979), pp. 60–69. ISSN: 0364-765X (cit. on p. 10).

[28] John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. "Trust Region Policy Optimization." In: *ICML*. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 1889–1897 (cit. on p. 25).

[29] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. "Policy Gradients with Parameter-Based Exploration for Control." In: *ICANN (1)*. Ed. by Vera Kurková, Roman Neruda, and Jan Koutník. Vol. 5163. Lecture Notes in Computer Science. Springer, 2008, pp. 387–396. ISBN: 978-3-540-87535-2 (cit. on p. 12).

[30] Richard S Sutton. "Learning to predict by the methods of temporal differences." In: *Machine learning* 3.1 (1988), pp. 9–44 (cit. on p. 10).

[31] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981 (cit. on pp. 5, 10, 17).

[32] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. Ed. by Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller. The MIT Press, 1999, pp. 1057–1063. ISBN: 0-262-19450-3 (cit. on pp. 1, 7, 12, 14, 17, 65).

[33] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. 1999, pp. 1057–1063 (cit. on p. 15).

[34] Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In: *Advances in Neural Information Processing Systems 12*. Ed. by S. A. Solla, T. K. Leen, and K. Müller. MIT Press, 2000, pp. 1057–1063 (cit. on p. 62).

[35] Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. "High confidence policy improvement." In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015, pp. 2380–2388 (cit. on p. 26).

[36] John N. Tsitsiklis. "On the Convergence of Optimistic Policy Iteration." In: *Journal of Machine Learning Research* 3 (2002), pp. 59–72 (cit. on p. 10).

[37] Paul Wagner. "A reinterpretation of the policy oscillation phenomenon in approximate policy iteration." In: *Advances in Neural Information Processing Systems 24*. 2011, pp. 2573–2581 (cit. on p. 19).

[38] Christopher JCH Watkins and Peter Dayan. "Q-learning." In: *Machine learning* 8.3 (1992), pp. 279–292 (cit. on p. 10).

[39] George Weiss. "Dynamic Programming and Markov Processes. Ronald A. Howard. Technology Press and Wiley, New York, 1960. viii + 136 pp. Illus." In: *Science* 132.3428 (1960), pp. 667–667. ISSN: 0036-8075 (cit. on p. 7).

[40] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Machine Learning* 8.3 (1992), pp. 229–256. ISSN: 1573-0565 (cit. on pp. 14, 15, 36, 65).

[41] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. "Analysis and Improvement of Policy Gradient Estimation." In: *NIPS*. Ed. by John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger. 2011, pp. 262–270 (cit. on p. 16).

[42] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. "Analysis and improvement of policy gradient estimation." In: *Neural Networks* 26 (2012), pp. 118–129 (cit. on pp. 36, 37).

A

PROOFS

In this appendix we provide the proofs that were omitted in Chapter 4. Some auxiliary lemmas are also introduced and proved.

Lemma A.0.1. *For any initial state distribution μ and any pair of stationary Gaussian policies $\pi_\theta \sim \mathcal{N}(\theta^T \boldsymbol{\phi}(s), \sigma^2)$ and $\pi_{\theta'} \sim \mathcal{N}(\theta'^T \boldsymbol{\phi}(s), \sigma^2)$, so that $\theta' = \theta + \Lambda \nabla_\theta J_\mu(\theta)$, and for any state s and action a:*

$$\pi_{\theta'}(a|s) - \pi_\theta(a|s) \geqslant \nabla_\theta \pi_\theta(a|s)^T \Lambda \nabla_\theta J_\mu(\theta) - \frac{M_\phi^2 \, \|\Lambda \nabla_\theta J_\mu(\theta)\|_1^2}{\sqrt{2\pi}\sigma^3}.$$

Proof. By exploiting Taylor's expansion

$$\begin{aligned}\pi_{\theta'}(a|s) =& \pi_{\theta + \Lambda \nabla_\theta J_\mu(\theta)}(a|s) = \\ & \pi_\theta(a|s) + \nabla_\theta \pi_\theta(a|s)^T \Delta\theta + R_1(\Delta\theta),\end{aligned}$$

where $R_1(\Delta\theta)$ is the remainder of the series, which can be bounded as follows by exploiting Lemma 4.1 from [24] and Assumption 3.1:

$$\begin{aligned}R_1(\Delta\theta) &= \sum_{i=1}^{m}\sum_{j=1}^{m} \frac{\partial^2 \pi_\theta(a|s)}{\partial\theta_i \partial\theta_j}\bigg|_{\theta + c\Delta\theta} \frac{\Delta\theta_i \Delta\theta_j}{1 + I(i=j)} && \text{for some } c \in (0,1) \\ &\geqslant -\sum_{i=1}^{m}\sum_{j=1}^{m} \frac{|\phi_i(s)\phi_j(s)|}{\sqrt{2\pi}\sigma^3} \frac{\Delta\theta_i \Delta\theta_j}{1 + I(i=j)} \\ &= -\frac{1}{\sqrt{2\pi}\sigma^3} \sum_{i=1}^{m}\sum_{j=1}^{m} \frac{\alpha_i |\phi_i(s)| \nabla_{\theta_i} J_\mu(\theta) \alpha_j |\phi_j(s)| \nabla_{\theta_j} J_\mu(\theta)}{1 + I(i=j)} \\ &= -\frac{(|\nabla_\theta J_\mu(\theta)|^T \Lambda |\boldsymbol{\phi}(s)|)^2}{\sqrt{2\pi}\sigma^3} \\ &\geqslant -\frac{M_\phi^2 \, \|\Lambda \nabla_\theta J_\mu(\theta)\|_1^2}{\sqrt{2\pi}\sigma^3}.\end{aligned}$$

It is now enough to apply this bound to Taylor's expansion. ∎

Lemma A.0.2. *For any initial state distribution μ and any pair of stationary Gaussian policies $\pi_\theta \sim \mathcal{N}(\theta^T \boldsymbol{\phi}(s), \sigma^2)$ and $\pi_{\theta'} \sim \mathcal{N}(\theta'^T \boldsymbol{\phi}(s), \sigma^2)$, so that $\theta' = \theta + \Lambda \nabla_\theta J_\mu(\theta)$:*

$$\|\pi_{\theta'} - \pi_\theta\|_\infty^2 \leqslant \frac{M_\phi^2 \, \|\Lambda \nabla_\theta J_\mu(\theta)\|_1^2}{\sigma^2}.$$

Proof. By exploiting Pinsker's inequality [22]

$$\begin{aligned}\|\pi_{\theta'} - \pi_\theta\|_\infty^2 &= \sup_s \|\pi_{\theta'} - \pi_\theta\|_\infty^2 \\ &\geqslant \sup_s 2H(\pi_{\theta'} \| \pi_\theta) \\ &= \sup_s \frac{1}{\sigma^2} \sum_i (\nabla_{\theta_i} J_\mu(\theta) \alpha_i \phi_i(s))^2 \\ &\geqslant \frac{M_\phi^2 \, \|\Lambda \nabla_\theta J_\mu(\theta)\|_1^2}{\sigma^2},\end{aligned}$$

where $H(P\|Q)$ is the Kullback-Liebler divergence. ■

Lemma 4.2.2. *For any initial state distribution μ and any pair of stationary Gaussian policies $\pi_\theta \sim \mathcal{N}(\theta^T \boldsymbol{\phi}(s), \sigma^2)$ and $\pi_{\theta'} \sim \mathcal{N}(\theta'^T \boldsymbol{\phi}(s), \sigma^2)$, so that $\theta' = \theta + \Lambda \nabla_\theta J_\mu(\theta)$, and under Assumption 4.2.1, the difference between the performance of $\pi_{\theta'}$ and the one of π_θ can be bounded below as follows:*

$$\begin{aligned}J_\mu(\theta') - J_\mu(\theta) \geqslant\ & \nabla_\theta J_\mu(\theta)^T \Lambda \nabla_\theta J_\mu(\theta) \\ & - \frac{\|\Lambda \nabla_\theta J_\mu(\theta)\|_1^2 \, M_\phi^2}{(1-\gamma)\sigma^2} \\ & \left(\frac{1}{\sqrt{2\pi}\sigma} \int_S d_\mu^{\pi_\theta}(s) \int_A Q^{\pi_\theta}(s,a) da ds + \frac{\gamma \, \|Q^{\pi_\theta}\|_\infty}{2(1-\gamma)} \right),\end{aligned}$$

where $\|Q^{\pi_\theta}\|_\infty$ is the supremum norm of the Q-function:

$$\|Q^{\pi_\theta}\|_\infty = \sup_{s \in \mathcal{S}, a \in \mathcal{A}} Q^{\pi_\theta}(s,a).$$

Proof. We plug the results of Lemmas A.0.1 and A.0.2 into Lemma 3.1 from [24]:

$$\begin{aligned}J_\mu(\theta') - J_\mu(\theta) \geqslant & \frac{1}{1-\gamma} \int_S d_\mu^{\pi_\theta}(s) \int_A (\pi_{\theta'}(a|s) - \pi_\theta(a|s)) Q^{\pi_\theta}(s,a) da ds \\ & - \frac{\gamma}{2(1-\gamma)^2} \|\pi_{\theta'} - \pi_\theta\|_\infty^2 \, \|Q^{\pi_\theta}\|_\infty \\ \geqslant & \frac{1}{1-\gamma} \int_S d_\mu^{\pi_\theta}(s) \int_A \nabla_\theta \pi_\theta(a|s)^T \Lambda \nabla_\theta J_\mu(\theta) Q^{\pi_\theta}(s,a) da ds \quad (1) \\ & - \frac{M_\phi^2 \, \|\Lambda \nabla_\theta J_\mu(\theta)\|_1^2}{(1-\gamma)\sqrt{2\pi}\sigma^3} \int_S d_\mu^{\pi_\theta}(s) \int_A Q^{\pi_\theta}(s,a) da ds \quad (2) \\ & - \frac{\gamma M_\phi^2 \, \|\Lambda \nabla_\theta J_\mu(\theta)\|_1^2}{2(1-\gamma)^2 \sigma^2} \|Q^{\pi_\theta}\|_\infty . \quad (3)\end{aligned}$$

Term (1) can be simplified by using the Policy Gradient Theorem [34]:

$$\begin{aligned}& \frac{1}{1-\gamma} \int_S d_\mu^{\pi_\theta}(s) \int_A \nabla_\theta \pi(a|s,\theta)^T \Lambda \nabla_\theta J_\mu(\theta) Q^{\pi_\theta}(s,a) da ds \\ &= \frac{\nabla_\theta J_\mu(\theta)^T \Lambda}{1-\gamma} \int_S d_\mu^{\pi_\theta}(s) \int_A \nabla_\theta \pi(a|s,\theta) Q^{\pi_\theta}(s,a) da ds \\ &= \nabla_\theta J_\mu(\theta)^T \Lambda \nabla_\theta J_\mu(\theta).\end{aligned}$$

The proof now follows simply by rearranging terms. ■

Theorem 4.2.3. *For any initial state distribution μ and any pair of stationary Gaussian policies $\pi_\theta \sim \mathcal{N}(\theta^T \boldsymbol{\phi}(s), \sigma^2)$ and $\pi_{\theta'} \sim \mathcal{N}(\theta'^T \boldsymbol{\phi}(s), \sigma^2)$, so that $\theta' = \theta + \Lambda \nabla_\theta J_\mu(\theta)$, and under Assumption 4.2.1, the difference between the performance of $\pi_{\theta'}$ and the one of π_θ can be bounded below as follows:*

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \nabla_\theta J_\mu(\theta)^T \Lambda \nabla_\theta J_\mu(\theta) - c \left\| \Lambda \nabla_\theta J_\mu(\theta) \right\|_1^2,$$

where $c = \frac{R M_\phi^2}{(1-\gamma)^2 \sigma^2} \left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)} \right)$ *and $|\mathcal{A}|$ is the volume of the action space.*

Proof. For every state $s \in \mathcal{S}$ and every action $a \in \mathcal{A}$, the Q-function belongs to $\left[-\frac{R}{1-\gamma}, \frac{R}{1-\gamma} \right]$. As a consequence, $\int_{\mathcal{A}} Q^{\pi_\theta}(s,a) da \leqslant \frac{|\mathcal{A}| R}{1-\gamma}$ and $\|Q^{\pi_\theta}\|_\infty \leqslant \frac{R}{1-\gamma}$. The proof follows from applying these bounds to the expression of Lemma 4.2.2. ■

Theorem 4.2.6. *Under the same assumptions of Theorem 4.2.3, and provided that a policy gradient estimate $\hat{\nabla}_\theta J_\mu(\theta)$ is available, so that $\mathbb{P}(|\nabla_{\theta_i} J_\mu(\theta) - \hat{\nabla}_{\theta_i} J_\mu(\theta)| \geqslant \epsilon_i) \leqslant \delta, \forall i = 1, \ldots, m$, the difference between the performance of $\pi_{\theta'}$ and the one of π_θ can be bounded below with probability at least $(1-\delta)^m$ as follows:*

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \underline{\hat{\nabla}_\theta J_\mu}(\theta)^T \Lambda \underline{\hat{\nabla}_\theta J_\mu}(\theta) - c \left\| \Lambda \overline{\hat{\nabla}_\theta J_\mu}(\theta) \right\|_1^2,$$

where c is defined as in Theorem 4.2.3.

Proof. The proof immediately follows from Theorem 3.3 and the definitions of $\underline{\hat{\nabla}_\theta J_\mu}(\theta)$ and $\overline{\hat{\nabla}_\theta J_\mu}(\theta)$ provided in Section 4.2.2. Note that the saturation to 0 in $\underline{\hat{\nabla}_\theta J_\mu}(\theta)$ is necessary since $\nabla_\theta J_\mu(\theta)$ is taken with absolute value in the negative term of the original bound. ■

Theorem 4.3.2. *Under the hypotheses of Theorem 4.2.3 and Assumption 4.3.1, the cost-sensitive performance improvement measure Υ_δ, as defined in Definition 4.1, is maximized by the following step size and batch size:*

$$\alpha_k^* = \begin{cases} \frac{(13-3\sqrt{17})}{4c} & \text{if } k = \arg\max_i |\hat{\nabla}_{\theta_i} J_\mu(\theta)|, \\ 0 & \text{otherwise}, \end{cases}$$

$$N^* = \left\lceil \frac{(13+3\sqrt{17}) d_\delta^2}{2 \left\| \hat{\nabla}_\theta J_\mu(\theta) \right\|_\infty^2} \right\rceil,$$

where $c = \frac{R M_\phi^2}{(1-\gamma)^2 \sigma^2} \left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)} \right)$, *which guarantee with probability $(1-\delta)^m$ a performance improvement of:*

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \frac{393 - 95\sqrt{17}}{8} \left\| \hat{\nabla}_\theta J_\mu(\theta) \right\|_\infty^2 \geqslant 0.16 \left\| \hat{\nabla}_\theta J_\mu(\theta) \right\|_\infty^2.$$

Proof. We first optimize the cost function Υ_δ w.r.t Λ. Since Υ_δ is just the bound from Theorem 4.2.6 divided by N, we can use the result from Corollary 4.2.8, which under Assumption 4.3.1 can be expressed as:

$$\alpha_k^* = \begin{cases} \frac{\left(\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty - d_\delta/\sqrt{N}\right)^2}{2c\left(\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty + d_\delta/\sqrt{N}\right)^2} & \text{if } k = \arg\max_i |\hat{\nabla}_{\theta_i} J_\mu(\theta)|, \\ 0 & \text{otherwise,} \end{cases}$$

which yields:

$$\Upsilon_\delta(\Lambda^*, N) = \frac{\left(\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty - d_\delta/\sqrt{N}\right)^4}{4c\left(\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty + d_\delta/\sqrt{N}\right)^2 N}.$$

To justify the use of Corollary 4.2.8, our N^* must be compliant with 4.2.7, which, under Assumption 4.3.1, translates into the following constraint:

$$N \geqslant N_0 := \frac{d_\delta^2}{\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty^2}.$$

By computing the derivative $\partial\Upsilon_\delta/\partial N$ we find just two stationary points in $[N_0, +\infty)$: the first one is N_0 itself, which is a minimum, since $\Upsilon_\delta(\Lambda^*, N_0) = 0$ and Υ_δ is non-negative. The other stationary point is our optimal batch size:

$$N^* = \frac{(13 + 3\sqrt{17})d_\delta^2}{2\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty^2}.$$

Since $\Upsilon_\delta(\Lambda^*, N_1) = 0$, $\lim_{N\to+\infty} \Upsilon_\delta(\Lambda^*, N) = 0$, and Υ_δ is continuous and differentiable in $[N_0, +\infty)$, N^* is indeed the global maximum in the region of interest. We can now substitute N^* into Λ^* to obtain:

$$\alpha_k^* = \begin{cases} \frac{(13-3\sqrt{17})}{4c} & \text{if } k = \arg\max_i |\hat{\nabla}_{\theta_i} J_\mu(\theta)|, \\ 0 & \text{otherwise,} \end{cases}$$

and into $\Upsilon_\delta(\Lambda^*, N^*)$ to obtain:

$$\Upsilon^* = \frac{(4977 - 1207\sqrt{17})\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty^4}{32 d_\delta^2}.$$

Finally

$$J_\mu(\theta') - J_\mu(\theta) \geqslant N^*\Upsilon^* = \frac{393 - 95\sqrt{17}}{8}\left\|\hat{\nabla}_\theta J_\mu(\theta)\right\|_\infty^2,$$

with probability at least $(1-\delta)^m$.

■

Lemma 4.3.3. *For any Gaussian policy* $\pi_\theta \sim \mathcal{N}(\theta^T \boldsymbol{\phi}(s), \sigma^2)$*, assuming that the action space is bounded* ($|a| \leqslant \overline{A}\ \forall a \in \mathcal{A}$) *and the policy gradient is estimated on trajectories of length* H*, the range* $\mathbf{R}$ *of the policy gradient sample* $\tilde{\nabla}_{\theta_i} J_\mu(\theta)$ *can be upper bounded* $\forall i = 1, \dots, m$ *and* $\forall \theta$ *by*

$$\mathbf{R} \leqslant \frac{2HM_\phi \overline{A} R}{\sigma^2(1-\gamma)}.$$

Proof. We focus on the REINFORCE[40] gradient estimator:

$$\hat{\nabla}_\theta J_\mu^{RF}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \left(\sum_{k=1}^{H} \nabla_\theta \log \pi(a_k^n | s_k^n, \theta) \left(\sum_{l=1}^{H} \gamma^{l-1} r_l^n - b \right) \right),$$

and the G(PO)MDP[3]/PGT[32] gradient estimator:

$$\hat{\nabla}_\theta J_\mu^{PGT}(\theta) = \hat{\nabla}_\theta J_\mu^{G(PO)MDP}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \left(\sum_{k=1}^{H} \nabla_\theta \log \pi(a_k^n | s_k^n, \theta) \left(\sum_{l=k}^{H} \gamma^{l-1} r_l^n - b_l^n \right) \right).$$

In both cases, the i-th component of the single sample can be bounded in absolute value as

$$\sum_{k=1}^{H} \nabla_{\theta_i} |\log \pi(a_k^n | s_k^n, \theta)| \frac{R}{1-\gamma}.$$

In the case of Gaussian policy, bounded action space ($|a| \leqslant \overline{A}$) and under Assumption 4.2.1, in the most general case, the range of the term $\nabla_{\theta_i} \log \pi(a_k^n | s_k^n, \theta)$ can be bounded as

$$\frac{2M_\phi \overline{A}}{\sigma^2}.$$

Finally

$$\mathbf{R} \leqslant \frac{2HM_\phi \overline{A} R}{\sigma^2(1-\gamma)}.$$

■

B

EXTENSION TO SOFTMAX POLICIES

In this appendix we show a first attempt at adapting the results on monotonic improvement to the class of Softmax Policies, as suggested in Chapter 6. The impractical nature of this partial results is indicative of the difficulties that characterize this task and could be useful to guide future research on the matter.

We fist need some Lemmas about specific properties of Softmax policies:

Lemma B.0.3. *If π_θ is a softmax policy:*

$$\left|\frac{\partial^2 \pi_\theta(a \mid s)}{\partial\theta_i \partial\theta_j}\right| \leqslant 6M_\phi^2.$$

Proof.

$$\begin{aligned}\left|\frac{\partial^2 \pi_\theta(a \mid s)}{\partial\theta_i \partial\theta_j}\right| &= \Big|\pi_\theta(a \mid s)\Big(\phi_j(s,\cdot)\phi_i(s,\cdot) - \phi_i(s,\cdot)\mathop{\mathbf{E}}_{a'\sim\pi_\theta}\left[\phi_j(s,\cdot)\right] - \phi_j(s,\cdot)\mathop{\mathbf{E}}_{a'\sim\pi_\theta}\left[\phi_i(s,\cdot)\right] \\ &\qquad +2\mathop{\mathbf{E}}_{a'\sim\pi_\theta}\left[\phi_i(s,\cdot)\right]\mathop{\mathbf{E}}_{a'\sim\pi_\theta}\left[\phi_j(s,\cdot)\right] - \mathop{\mathbf{E}}_{a'\sim\pi_\theta}\left[\phi_i(s,\cdot)\phi_j(s,\cdot)\right]\Big)\Big| \\ &\leqslant 6M_\phi^2.\end{aligned}$$

■

Lemma B.0.4. *If π_θ is a softmax policy and $\theta' = \theta + \Lambda\nabla_\theta J_\mu(\theta)$:*

$$\pi_{\theta'}(a \mid s) - \pi_\theta(a \mid s) \geqslant \nabla_\theta \pi_\theta(a \mid s)^T \Lambda \nabla_\theta J_\mu(\theta) - 6M_\phi^2 \left\|\Lambda\nabla_\theta J_\mu(\theta)\right\|_1^2.$$

Proof. It follows from the application of Lemma B.0.3 to Taylor's expansion. ■

Lemma B.0.5. *If π_θ is a softmax policy and $\theta' = \theta + \Lambda\nabla_\theta J_\mu(\theta)$:*

$$\left\|\pi_{\theta'} - \pi_\theta\right\|_\infty^2 \leqslant 4M_\phi \left\|\Lambda\nabla_\theta J_\mu(\theta)\right\|_1.$$

Proof. By Pinsker's inequality:

$$\left\|\pi_{\theta'} - \pi_\theta\right\|_\infty^2 = \sup_s \left\|\pi_{\theta'} - \pi_\theta\right\|_1^2 \leqslant \sup_s \left(2H(\pi_{\theta'} \,\|\, \pi_\theta)\right),$$

where $H(\cdot \,\|\, \cdot)$ is the Kullback-Liebler divergence, which in this case can be bounded as:

$$\begin{aligned}
H(\pi_{\theta'}, \pi_\theta) &= \sum_{a \in \mathcal{A}} \frac{e^{\theta'^T \phi(s,a)}}{\sum_{a' \in \mathcal{A}} e^{\theta'^T \phi(s,a')}} \log \frac{e^{\theta'^T \phi(s,a)} \sum_{a' \in \mathcal{A}} e^{\theta^T \phi(s,a')}}{e^{\theta^T \phi(s,a)} \sum_{a' \in \mathcal{A}} e^{\theta'^T \phi(s,a')}} \\
&= \mathop{\mathbf{E}}_{a \sim \pi_{\theta'}} \left[\Delta\theta^T \phi(s,a) + \log \frac{\sum_{a' \in \mathcal{A}} e^{\theta^T \phi(s,a')}}{\sum_{a' \in \mathcal{A}} e^{\theta'^T \phi(s,a')}} \right] \\
&= \mathop{\mathbf{E}}_{a \sim \pi_{\theta'}} \left[\Delta\theta^T \phi(s,a) + \log \frac{\sum_{a' \in \mathcal{A}} e^{\theta^T \phi(s,a')}}{\sum_{a' \in \mathcal{A}} e^{\theta^T \phi(s,a')} e^{\Delta\theta^T \phi(s,a')}} \right] \\
&\leqslant \mathop{\mathbf{E}}_{a \sim \pi_{\theta'}} \left[\|\Delta\theta\|_1 M_\phi + \log \frac{\cancel{\sum_{a' \in \mathcal{A}} e^{\theta^T \phi(s,a')}}}{e^{-\|\Delta\theta\|_1 M_\phi} \cancel{\sum_{a' \in \mathcal{A}} e^{\theta^T \phi(s,a')}}} \right] \\
&= 2 \|\Delta\theta\|_1 M_\phi = 2 M_\phi \|\Lambda \nabla_\theta J_\mu(\theta)\|_1 .
\end{aligned}$$

The rest of the proof is trivial. ■

We can now state the following theorem, which is a variant of Theorem 4.2.3

Theorem B.0.6. *For any initial state distribution μ, if π_θ is a softmax policy and $\theta' = \theta + \Lambda \nabla_\theta J_\mu(\theta)$:*

$$\begin{aligned}
J_\mu(\theta') - J_\mu(\theta) \geqslant\ & \nabla_\theta J_\mu(\theta)^T \Lambda \nabla_\theta J_\mu(\theta) \\
& - c \|\Lambda \nabla_\theta J_\mu(\theta)\|_1^2 - b \|\Lambda \nabla_\theta J_\mu(\theta)\|_1 ,
\end{aligned}$$

where

$$\begin{aligned}
c &= \frac{6 M_\phi^2 |\mathcal{A}| R}{(1-\gamma)^2}, \\
b &= \frac{2\gamma M_\phi R}{(1-\gamma)^3}.
\end{aligned}$$

Proof. It's enough to plug the results from Lemmas B.0.4 and B.0.3 into Lemma 3.1 from [24]. ■

Corollary B.0.7. *The performance lower bound of Theorem B.0.6 is maximized, under the assumption that $b \leqslant \|\nabla_\theta J_\mu(\theta)\|_\infty$, by the following non-scalar step size:*

$$\alpha_k^* = \begin{cases} \frac{1}{2c} \left(1 - \frac{b}{\|\nabla_\theta J_\mu(\theta)\|_\infty} \right) & \text{if } k = \arg\max_i |\nabla_{\theta_i} J_\mu(\theta)|, \\ 0 & \text{otherwise,} \end{cases}$$

which guarantees the following performance improvement:

$$J_\mu(\theta') - J_\mu(\theta) \geqslant \frac{\left(\|\nabla_\theta J_\mu(\theta)\|_\infty - b \right)^2}{4c}.$$

Proof. The proof is very similar to the one of Corollary 4.2.4, so we omit it. ■

The issue with the result of Corollary B.0.7 is that, in practical applications, the constant b can easily be larger than $\|\nabla_\theta J_\mu(\theta)\|_\infty$. This means that, in many cases, no improvement-guaranteeing step size can be found, not even when the policy gradient is known exactly. Although this result cannot be used in practice, it shows that the main difficulty in extending our method to Softmax policies is the characterization of the policy difference measure $\|\pi_{\theta'} - \pi_\theta\|_\infty^2$. Further research should be aimed at overcoming or bypassing this issue.