POLITECNICO DI MILANO

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

Master of Science in Computer Science and Engineering

Department of Electronics, Information and Bioengineering

Balancing Safety and Exploration in Policy Gradient

AI & R Lab
Artificial Intelligence & Robotics Lab
Politecnico di Milano

Supervisor: Prof. Marcello Restelli
Co-supervisor: Dott. Matteo Papini

Author:
Andrea Battistello, 873795

Academic Year 2017-2018

Alla mia famiglia

Abstract

Reinforcement Learning is a powerful framework that can be used to solve complex control tasks. Among the current challenges that reinforcement learning has to face, there are the inner difficulties of exploring the environment and doing it safely. Safe Reinforcement Learning is necessary for critical applications, such as robotics, where exploratory behaviours can harm systems and people, but it also lends itself to economic interpretations. However, safe algorithms often tend to be overly conservative, sacrificing too much in terms of speed and exploration. The latter, in particular, is of fundamental importance for a learning algorithm to gain informations about the environment. In this thesis, we will investigate the non-trivial tradeoff between these two competing aspects, safety and exploration. Starting from the idea that a practical algorithm should be safe as needed, but no more, we identify interesting application scenarios and propose Safely-Exploring Policy Gradient (SEPG), a very general policy gradient framework that can be customized to match particular safety constraints. To do so, we generalize existing bounds on performance improvement for Gaussian policies to the adaptive-variance case and propose policy updates that are both safe and exploratory.

Estratto in lingua italiana

Ogni giorno dobbiamo compiere delle scelte: scegliamo quali vestiti indossare, se è più conveniente prendere l'autobus o la metropolitana, se impostare la sveglia per il giorno successivo e così via. Abbiamo imparato come affrontare questi problemi da quando eravamo bambini. Il processo che ci ha portato ad acquisire questa capacità è lungo e complesso, ma può essere suddiviso in pochi elementi essenziali: i) abbiamo una percezione di ciò che ci circonda, ii) possiamo interagire con l'ambiente compiendo delle azioni e iii) l'ambiente reagisce dandoci una nuova percezione. Questi tre fattori entrano sempre in gioco quando vogliamo apprendere una nuova abilità (e.g., giocare a tennis). L'interazione con ciò che ci circonda costituisce ciò che chiamiamo esperienza. Tuttavia, l'esperienza da sola non è sufficiente: dobbiamo anche identificare un obiettivo. In termini molto semplici, possiamo dire di essere felici se l'obiettivo è stato raggiunto, altrimenti siamo tristi. La quantità di felicità (o tristezza) che otteniamo può essere vista come una ricompensa che l'ambiente ci offre. Ciò che istintivamente impariamo a fare è sviluppare delle strategie che aumentino la quantità di felicità ottenuta tramite la nostra interazione con l'ambiente. Un esempio molto calzante è dato dal processo di addestramento dei cani: il cane impara a stare seduto perché sa che poi riceverà una ricompensa.
Questo processo di apprendimento, quando applicato a programmi informatici, prende il nome di apprendimento per rinforzo.

L'apprendimento per rinforzo (RL) è un campo dell'intelligenza artificiale che imita il modo in cui gli esseri viventi apprendono in natura. Questi algoritmi di apprendimento si basano su un modello matematico semplificato, ma potente, che è in grado di descrivere le interazioni tra un agente e l'ambiente che lo circonda. Il risultato finale è una funzione che associa, ad ogni possible situazione, l'azione che l'agente deve compiere per massimizzare la ricompensa che potrà ricevere.

Queste ricompense hanno lo scopo di fornire informazioni su *quale* è l'obiettivo da perseguire ma non su *come* fare per raggiungerlo. All'agente non viene detto quali azioni intraprendere, piuttosto, deve scoprire da solo quali azioni sono le più redditizie. L'agente può imparare il valore delle proprie azioni procedendo per tentativi e accumulando esperienza.
Questo non è un compito banale, dal momento che le azioni possono potenzialmente avere effetti a lungo termine. Infatti, un'azione influisce sullo stato successivo, da cui un'altra azione può condurre a un altro stato ancora e così via, generando una

sequenza di eventi causalmente correlati. Le ricompense a lungo termine e la ricerca per tentativi sono i due aspetti più importanti dell'apprendimento per rinforzo, che rendono questo problema difficile ma intrigante.

Tra le sfide attuali nell'ambito dell'apprendimento per rinforzo, compare il problema di come esplorare l'ambiente e di come farlo in modo sicuro. L'esplorazione è un aspetto fondamentale dell'apprendimento, poiché risolvere efficacemente un compito difficile richiede anche creatività. In questo contesto, esplorare significa come scegliere azioni che sono diverse da quelle a cui l'agente è abituato. Cambiare leggermente il modo in cui facciamo qualcosa può aiutarci ad acquisire una maggiore conoscenza dell'ambiente e, potenzialmente, aiutarci a trovare nuove soluzioni. Tuttavia, un eccesso di esplorazione non è sempre utile e può danneggiare macchine e persone, oltre ad avere un possibile impatto economico. Ad esempio, possiamo pensare a un braccio robotico che sta imparando a colpire una palla con una mazza da baseball. Può accadere che, durante la fase di apprendimento, il braccio robotico agiti la mazza con movimenti bruschi, i quali potrebbero potenzialmente portare ad una rottura. Questo problema si riflette economicamente anche nei costi di riparazione del braccio robotico.

In questa tesi vogliamo combinare questi due aspetti contrastanti di sicurezza durante l'apprendimento ed esplorazione, ed escogitare una soluzione generale che possa essere adattata alle esigenze dell'utente. Ci concentreremo su una particolare classe di algoritmi di apprendimento, chiamata 'policy gradient' [1]. Oltre ad avere buone proprietà di convergenza, questi metodi sono particolarmente adatti per i problemi di controllo con azioni continue, in quanto sono in grado di risolvere molte delle difficoltà che si presentano in questo campo, come lo spazio di stati multidimensionale e sensori non affidabili. Inoltre, consentono di integrare facilmente delle conoscenze pregresse sul problema, al fine di progettare soluzioni più sicure ed efficaci. Il lavoro fatto recentemente nel campo dell'apprendimento per rinforzo sicuro ha portato a buoni risultati teorici. In particolare faremo riferimento al lavoro di Kakade e Langford [2] su 'conservative policy iteration', successivamente perfezionato da Pirotta et al. [3] e adattato al metodo policy gradient in [4]. Tuttavia, questi risultati non considerano il fattore esplorativo, che è di grande importanza nell'apprendimento. Inoltre, fanno riferimento al solo caso, limitato, di miglioramenti monotoni, in cui l'obiettivo principale è migliorare la soluzione corrente ad ogni iterazione. Questo vincolo ha un forte impatto sulla velocità di apprendimento, dal momento che dobbiamo fare aggiornamenti molto piccoli per migliorare in modo sicuro la soluzione corrente. Per questo motivo, i metodi introdotti precedentemente sono noti per essere eccessivamente cauti, in quanto sacrificano troppo in termini di velocità di apprendimento ed esplorazione.

Partendo da questi risultati, introdurremo un nuovo framework per l'apprendimento per rinforzo sicuro. Questo framework va oltre il singolo caso di miglioramento monotono, e include diversi tipi di vincoli, ciascuno corrispondente a una diversa esigenza pratica. L'utente, quindi, potrà scegliere il vincolo di sicurezza che meglio si adatta alle sue esigenze, senza garantire più di quanto sia necessario. All'interno di questo framework, estenderemo i risultati di Pirotta et al. [4] per occuparci anche dell'esplorazione adattiva. Il contributo finale che presenteremo in questo lavoro è un algoritmo generale denominato Safely-Exploring Policy Gradient (SEPG), che combina i nuovi risultati sviluppati in questa tesi. SEPG può essere personalizzato per soddisfare i diversi vincoli di sicurezza descritti nel nuovo framework appena introdotto.

Ringraziamenti

Vorrei ringraziare innanzitutto la mia famiglia, che mi ha sempre sostenuto in questo lungo percorso, sia a livello economico che a livello morale, insegnandomi a dare sempre il massimo per inseguire le proprie passioni.

Fondamentale è stato l'aiuto e il supporto del prof. Marcello Restelli e del Dott. Matteo Papini, a cui dedico un ringraziamento speciale per il loro insostituibile supporto nella stesura di questa tesi. A loro, che attraverso l'insegnamento riescono a trasmettere la loro passione per la ricerca, sono profondamente grato.

Infine, ma non meno importanti, ringrazio i miei amici e compagni di corso, che sono stati fondamentali nei momenti di svago e di studio.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Every day we are required to make some choices: we choose what to wear everyday, if it is more convenient to take the bus or the metro, whether to put the alarm for the next day and so on. We, as decision-makers, have learned how to tackle these problems since we were children. The process that brought us to acquire this ability is long and complex, but it can be broken down to few essential pieces: i) we have a perception of what is surrounding us, ii) we can interact with the environment by taking some actions and iii) the environment reacts by giving us a new perception. These three factors always come into play, whether we want to learn a new skill (e.g., to play the piano), we want to play a game and so on. This interaction with the environment builds what we call experience. However, experience alone is not enough: we also have to identify a goal. In a very simple setting, we can say that we are happy if we accomplish the goal, otherwise we are sad. The amount of happiness (or sadness) can be seen as a reward that the environment gives us. What we naturally learn to do is to develop strategies that increase the amount of happiness obtained by our interaction with the environment. This process of learning, when applied to computer programs, takes the name of reinforcement learning.

Reinforcement Learning (RL) is a field of Machine Learning that closely mimics how living beings learn in nature. RL algorithms are based on a simplified yet powerful mathematical model, that is able to describe the interactions between an agent and the environment. The final result will be a function that maps states to actions, telling the agent what to do, so as to maximize the reward that it is able to gain. This reward signal is intended to give insights about *what* is the goal to pursue, but not *how*. The learner is not told which actions to take, but rather it has to discover which actions are more valuable via trial-and-error, from experience.
This is not a trivial task, since actions can potentially have long-term effects. As a matter of facts, an action affects the following state, from where another action can lead to another state and so on, generating a sequence of causally related events. Delayed rewards and trial-and-error search are the two most important aspects of reinforcement learning, that make this problem both difficult and intriguing.

1.1 Motivation

Among the current challenges that reinforcement learning has to face, there are the inner difficulties of exploring the environment and how to do it safely. Exploration is a fundamental aspect of learning, as it requires creativity and practice to effectively solve a difficult task. In this context, exploration is intended as taking actions that are different than the ones we are used to take. Slightly changing the way we do something can help us gain more knowledge about the environment, and potentially can help find new solutions. However, an excess in exploration is not always useful and can harm systems and people, besides having an intuitive economical impact. As an example, we can think of a robotic arm that is learning how to hit a ball with a baseball bat. It can happen that, during the learning phase, the robotic arm swings the bat with sudden movements, that could potentially break the system. This problem is also reflected in the costs of repairing the robotic arm.

1.2 Contribution

In this thesis, we want to combine the two competing aspects of safety and exploration, and devise a general solution that can be customized to the user needs. We will focus on a particular class of algorithms in the reinforcement learning literature called policy gradient methods [1]. Besides having nice convergence properties, these methods are particularly suited for control tasks with continuous actions, as they are able to address many of the difficulties that arise in this field, 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 results. Recent works in safe reinforcement learning have brought to nice theoretical results. In particular, we will refer to the work of Kakade and Langford [2] on conservative policy iteration, later refined by Pirotta et al. [3] and adapted to the policy gradient framework in [4]. However, these results do not take into consideration the exploration factor. Moreover, they mostly consider the restricted case of monotonic improvements, where the main goal is to improve the current solution at every iteration. This constraint has a big impact on the learning speed of the algorithm, since we need to take very small updates to safely improve the current solution. For this reason, these methods are known to be overly conservative, sacrificing too much in terms of speed and exploration.

Starting from these results, we will introduce a new framework for safe reinforcement learning. This framework goes beyond the monotonic improvement case, and will include several kinds of constraints, each corresponding to different practical need. The user can then choose the type of safety constraint that better suits his/her requirements, without guaranteeing more than what is needed. Within this framework, we will extend the results from [4] to also deal with adaptive exploration. The final contribution that we introduce in this work is a general algorithm named Safely-Exploring Policy Gradient (SEPG), that combines the new results developed in this thesis. SEPG can be customized to match several safety constraints described in the newly adopted framework.

1.3 Structure of the thesis

The contents of the thesis are organized as follows. In Chapter 2, we will provide an overview of reinforcement learning and we will introduce the basic concepts that are needed to define policy gradient methods. In particular we will define a mathematical model called Markov Decision Process that describes the interactions between the agent and the environment. In Chapter 3, we will describe in details the main results on safe reinforcement learning. Specifically, we will start from the results of Kakade and Langford [2] on conservative policy iteration, that were later extended by [3] with the introduction of a tighter bound on performance improvement. Pirotta et al. [4] adapted these results to policy gradient, that were later extended by Papini et al. [5] to also include adaptive batch sizes. In Chapter 4, we will introduce the main contribution of this thesis. We will first provide the new safe reinforcement learning framework where we identify two main classes of safety constraints. Then, we will derive new results for the adaptive exploration case and finally we will see how to include these results in a practical algorithm named Safely-Exploring Policy Gradient (SEPG). We will describe in details how to adapt SEPG to match every safety constraint. Next, in Chapter 5, we will evaluate SEPG variants on simulated continuous control tasks. We will also compare SEPG with other non-safe approaches and we will discuss the results. Finally, in Chapter 6, we will derive some conclusions and we will describe possible additions that can be studied in future works.

Chapter 2

Reinforcement Learning

In this chapter, we will formalize the problem of reinforcement learning using a model taken from dynamical systems theory called Markov decision process. This model is able to capture the main aspects that build an interaction between the agent and the environment: observations, actions and goal. This model will be described in details in Section 2.1. Next, three general methods to solve an MDP are presented in Section 2.2: policy iteration, value iteration and policy search. We will briefly describe the former two methods, enough to introduce a policy search method called Policy Gradient, that is the main interest of this work. Policy Gradient methods will then be described in detail in Section 2.3, including some aspects about gradient approximation, baselines and other more advanced refinements.

2.1 Markov Decision Processes

In this section we will give a formal definition of reinforcement learning by defining a mathematical model called Markov decision process. For a complete treatment of reinforcement learning, refer to [6].
The main aspects that we want to capture are the interactions between an agent and the environment over time. A learning agent must be able to sense the state of its environment and take some actions, while the environment itself receives the actions from the agent and reacts by changing the state and giving feedbacks to the agent, completing a cycle. The agent must also have a goal that is related to a state of the environment.
Markov decision processes are intended to include just these three aspects – sensation, action and goal – in their simplest possible forms without trivializing any of them.

Definition 2.1. A Markov Decision Process (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu \rangle$ where:

– $\mathcal{S}$ is the set of possible **states**

– $\mathcal{A}$ is the set of possible **actions**

– $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the **transition function** that, given the current state and current action, outputs a probability distribution over the next state.

– $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the **reward function** that maps a scalar reward to every state-action pair

– γ is the **discount factor**

– $\mu : \Delta(\mathcal{S})$ is a probability distribution over the **initial state**.

The dynamics of the environment, as represented by the agent, must satisfy two fundamental properties:

- Stationarity: the transition function $\mathcal{P}$ does not change over time.
- Markov property: the transition function $\mathcal{P}$ only depends on the previous state, not on the entire history of the interaction.

An agent in an MDP starts from an initial state drawn from the probability distribution μ, $s_0 \sim \mu$. From this state, the agent can interact with the environment by executing an action $a_0 \in \mathcal{A}$. The environment receives action a_0 and reacts to this stimulus by returning to the agent the next state s_1 drawn from the transition function $s_1 \sim \mathcal{P}(s_0, a_0)$ and a scalar reward $r_1 = \mathcal{R}(s_0, a_0)$. From its new observation, the agent can continue the interaction with the environment by choosing another action a_1 and so on until it reaches the goal. We define the sequence of collected states, actions and rewards $\tau \sim s_0, a_0, s_1, r_1, a_1, s_2, \dots$ as a trajectory. Sometimes we will refer to a trajectory with terms like 'sample' or 'experience', since the collected information will be used for the learning procedure.

Figure 2.4 shows a graphical representation of an MDP. The circles represent the states of the environment and the arrows represent the transition probability of going from s_t to s_{t+1}. Each state is associated with a reward, depicted in red.
Assuming that a student starts from Class 1, he/she can be distracted and use the phone to access Facebook with probability 0.5. If this happens, the environment will transition to the new state and the student will receive a reward of -1. From this new state it would be hard for the student to return to Class 1, because there is a 0.9 probability of continuing to use Facebook (and receiving a negative reward). The goal, in this simple example, is to go to sleep.
This mathematical model is very general and can be used also to model more complex problems.

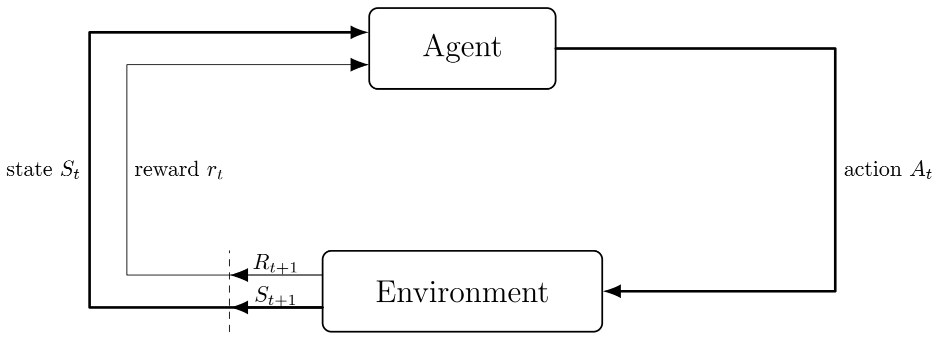


Figure 2.1: Agent-environment loop

Example 2.1 (Chess)**.** The game of chess can be modelled as an MDP with a discrete state space that encodes the position of all the pieces in the checkerboard, a discrete action space encoding all the possible moves of the pieces, a deterministic transition function that follows the rules of the game and a (possible) reward function:

$$\mathcal{R}(s,a) = \begin{cases} 1 & \text{Player wins} \\ -1 & \text{Player does not win} \end{cases}$$

Example 2.2 (Mountain Car)**.** A car is placed between two hills and can move left or right. The goal is to reach the top of the right hill, but the engine is not powerful enough to directly climb the hill, so the optimal solution is to gain momentum by first going in the opposite direction. This task can be modelled with a continuous state space $\mathcal{S} \in [-2, 2]$, a continuous action representing the force applied to the car, $\mathcal{A} \in [-1, 1]$, a deterministic transition function given by the laws of physics and the following reward function:

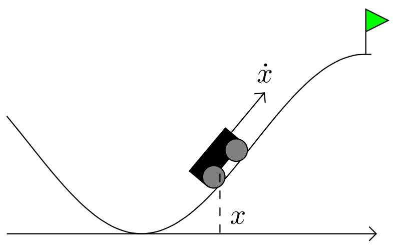


Figure 2.2: Mountain car task

$$\mathcal{R}(s,a) = \begin{cases} +100 & \text{when } s \geq 1.5 \\ -a^2 & \text{otherwise} \end{cases}$$

This problem is particularly difficult for its exploration needs: the agent is not aware of the presence of the goal until it is reached. Therefore, the agent needs to take actions that possibly have a bad impact on the immediate reward in order to reach the goal and being able to solve the problem. This concept of exploration (take actions to gain information about the environment) vs exploitation (take actions that you know are profitable) will be described in the next chapter.

Example 2.3 (Cartpole)**.** A pole is attached to a cart by a joint that is free to rotate. The cart moves along a frictionless track either left or right. The pole starts upright (with a small perturbation), and the goal is to prevent it from falling over by moving the cart left or right.

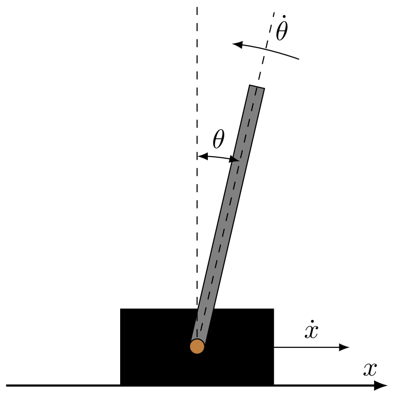


Figure 2.3: Cartpole task

This task can be modelled with a continuous multidimensional state space $\mathcal{S} \in \mathbb{R}^4$. The state features are: the position of the cart $s_0 \in [-2.4, 2.4]$, its velocity $s_1 \in [-\infty, \infty]$, the pole angle $s_2 \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and its angular velocity $s_3 \in [-\infty, \infty]$. The action space is continuous $\mathcal{A} \in [-1, 1]$ and represents the force to be applied on the cart. The transition function is given by the laws of physics and the reward is $+1$ for every step until the episode ends. An episode ends when a maximum step limit H is reached or when the pole is inclined more than 15 degrees from the vertical.

Intuitively, the agent aims at maximizing the sum of the collected rewards. However, this naïve sum of rewards does not apply if we consider infinite-horizon MDPs, which is a common setting in the literature. In fact, summing over potentially infinite-length trajectories will cause the cumulative reward to grow indefinitely and lose its original meaning. (e.g. earning $+1$ at every step will have the same meaning of earning $+100$ at every step, since over an infinite horizon they will both grow to infinity).
For this reason, it is common to consider a generalized version where we apply a discount factor γ to the immediate reward and consider the cumulative discounted reward:

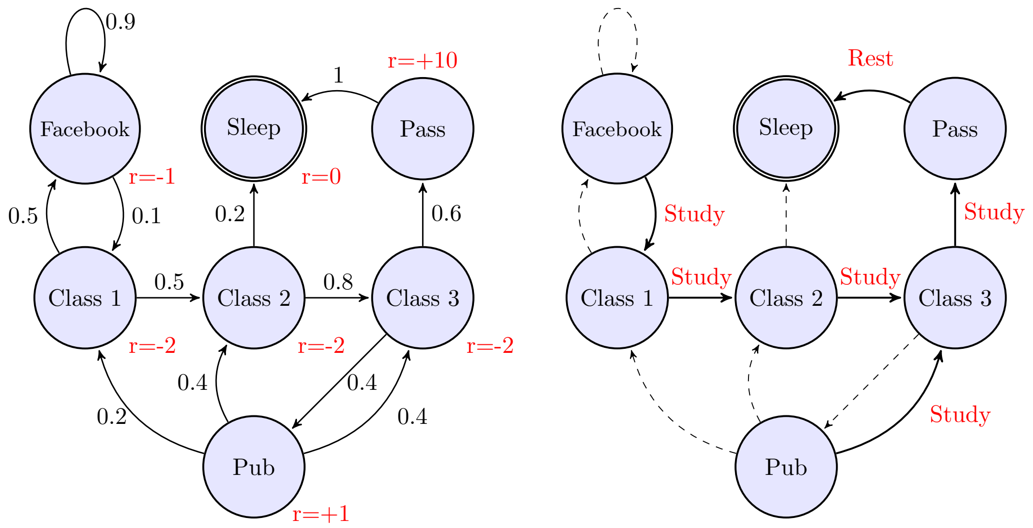
Definition 2.2 (Cumulative discounted reward)**.**

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

where $\gamma \in [0, 1]$ is called discount factor.

Note that the naïve cumulative reward can be obtained by setting $\gamma = 1$, and this is sometimes used when the horizon is finite.

However, the addition of the discount factor has a well-known two-fold meaning: at a first glance, it represents how the agent is interested in future rewards. With γ close to 0, we have a myopic behaviour, where we are only interested in immediate rewards rather than long-term investments. With γ close to 1, instead, we are giving more value to long-term rewards. In the second place, γ can also represent the probability that the simulation will continue for one more step. If γ is low, we should strive to get as much as we can because we have no guarantee that we could last one more step. The discount factor usually depends on the domain and the goal that we want to achieve: a long-term goal will require a value of $\gamma \simeq 1$.



(a) Visual representation of the transition function (black) and reward function (red)

(b) Visual representation of a (deterministic) policy

Figure 2.4: Representation of a simple MDP

The actual value of the discounted cumulative reward depends on the interaction of the agent with the environment over a trajectory. Many factors come into play to influence this objective function, most importantly the behaviour of the agent (i.e., which actions the agent chooses). The way an agent interacts with the environment, hence its behaviour, is defined by the policy function:

Definition 2.3. A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is a function that for each state $s \in \mathcal{S}$, outputs a probability distribution over the actions in $\mathcal{A}$. In other words, from state s the agent will choose action a with probability $\pi(a \mid s)$.

The policy is one of the main ingredients of reinforcement learning, as it is the final result of a reinforcement learning algorithm. The policy tells the agent which actions to choose in every state. Starting from an initial state s_0, the agent will choose action $a_0 \sim \pi(\cdot|s_0)$ drawn from the action distribution induced by the policy in this state.
Given an initial state and a policy, we can define its performance, or expected return, in this way:

Definition 2.4 (Performance of a policy π)**.** The performance (or expected return) of a policy π under initial state distribution μ is:

$$J_\mu(\pi) = \mathbb{E}_{\substack{a_k \sim \pi, s_0 \sim \mu \\ s_k \sim \mathcal{P}}} [R],$$

that is, the expected value of the discounted cumulative reward obtained by following the policy π starting from an initial state drawn from μ.

Having defined this new metric, we can easily rank each policy according to its performance: a policy π_1 is better than a policy π_0 if $J_\mu(\pi_1) > J_\mu(\pi_0)$. The next step will be to find the policy that achieves the highest performance for a given MDP. This is the goal of solving an MDP.

Problem formulation: To solve an MDP means to find an optimal policy π^* that maximises the average cumulative reward. More formally we want to find π^* such that $J_\mu(\pi^*) = \mathbb{E}_{a_k \sim \pi^*, s_0 \sim \mu}[R]$ is maximum.

Summing up, we can define the interaction between an agent and the environment as a sequence of state-action-reward tuples called a trajectory. The environment is modelled with a Markovian transition function - a function that takes the current state-action pair and outputs a probability distribution over the next state. The Markov property assures that the transition function depends only on the current state and not on the entire history of interactions. The agent receives a reward for every action it takes and aims at maximizing the discounted cumulative reward. The behaviour of the agent is characterized by the policy function π, which yields, for every state s, a probability distribution over actions $\pi(a|s)$. Every policy can be ranked by its performance or expected reward, obtained following the policy from an initial state drawn from μ. An MDP is considered solved when we find an optimal policy π^* whose performance value is maximal. For every stationary MDP, there exists an optimal policy that is deterministic, namely, a policy that for each state yields a single action with probability 1.

Here we define a new quantity that will be useful later which is the discounted future state distribution. This function calculates the visitation frequency for each state $s \in \mathcal{S}$ by starting from $s_0 \sim \mu$ and following policy π. This quantity is defined as follows:

Definition 2.5 (Future state distribution)**.** The (discounted) future state distribution is defined as:

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

This quantity identifies the expected state distribution starting from $s_0 \sim \mu$ and following policy π.

Following this definition, we can reformulate the performance of a policy π as a function of the state distribution $d_\mu^\pi(s)$.

Definition 2.6 (Performance given state distribution)**.** The performance of a policy π given an initial state distribution μ can be expressed as:

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

2.1.1 Value functions

So far we have defined what is an MDP and what it means to find an optimal policy π^*. In this section, we will take the problem under a different perspective by considering the value of each state. The value of a state should consider not only immediate rewards, but also the discounted sum of all the rewards that the agent may get. This idea of including delayed rewards to describe the value of a state is implemented by the state-value function:

Definition 2.7 (State-value function)**.** The value of a state s that can be obtained by running policy π is given by:

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

The state-value function $V^\pi(s)$ encodes the average reward that we will obtain starting from state s and following π thereafter. Note that this is a recursive definition, as to compute the value of state s we need to compute the values of all the states reachable from s and so on. This definition allows us to give a different way to calculate the performance of a policy based on the value function and the initial state distribution:

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

The value of the optimal policy π^* is called optimal state-value function $V^*(s) = V^{\pi^*}(s)$. The optimal value function is unique, even if multiple optimal policies may exist[1]. We can restate the goal of an RL algorithm as a maximisation over the policy space of the state-value function for all states:

$$\pi^* = \arg\max_{\pi \in \Pi} V^\pi(s), \qquad \forall s \in \mathcal{S}.$$

The value function is useful to evaluate policies, however it does not say anything about the actions that brought to that value. Since our main goal is to find an optimal policy, we need a way to assign a value also to the actions. The following statement will define the action-value function:

Definition 2.8 (Action-value function)**.** The action-value function for a policy π is defined as follows:

$$Q^\pi(s,a) = \mathcal{R}(s,a) + \gamma \int_{\mathcal{S}} \mathcal{P}(s' \mid s,a) \int_{\mathcal{A}} \pi(a' \mid s') Q^\pi(s',a') \, \mathrm{d}a' \, \mathrm{d}s'$$

The action-value function defines the value that we can obtain starting from state s, executing action a and following policy π thereafter. This function encodes more information, since we can retrieve the state-value function by averaging the action-value function over all actions: $V^\pi(s) = \mathbb{E}_{a \in \mathcal{A}}\left[Q^\pi(s,a)\right]$.
Similarly to the definition of the optimal value function, the optimal action-value function Q^* is defined to be the action-value function of the optimal policy: $Q^*(s) = Q^{\pi^*}$.

[1]An MDP can have multiple optimal policies, but what we are mainly concerned about is that at least one of them is deterministic and has the same value function of other optimal stochastic policies.

The action-value function allows us to introduce another quantity of particular interest for the algorithms described in Section 2.2, called the advantage function. The advantage of action a, state s, over policy π encodes how much we can gain by performing action a in state s instead of following the policy π, and is defined as follows:

Definition 2.9 (Advantage function)**.** The advantage function of action a over policy π in state s, is defined as:

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

The advantage function can also be used to compare two policies π' and π: for every state s we compute the advantage of selecting an action from π' rather than π. In this case, it is called policy advantage function.

Definition 2.10 (Policy advantage function)**.** The advantage of policy π' with respect to policy π in state s can be computed as:

$$A_{\pi}^{\pi'}(s) = \int_{\mathcal{A}} \pi'(a|s) A^{\pi}(s,a) \, \mathrm{d}a,$$

whose expected value over the states is called expected policy advantage function:

$$\mathbb{A}_{\pi,\mu}^{\pi'} = \int_{\mathcal{S}} d_{\mu}^{\pi}(s) A_{\pi}^{\pi'}(s) \, \mathrm{d}s.$$

2.2 General methods to solve an MDP

In the previous section we have seen that solving an MDP means finding the optimal policy π^* with the highest performance $J_{\mu}(\pi)$, which yields the optimal state-value function $V^*(s)$. Before diving into specific methods that aim at solving an MDP, we will provide some classification for learning algorithms.
First of all, we have to distinguish between a *target* policy and a *behaviour* policy: the target policy is the output of the algorithm, the one that is intended to solve the MDP. The behaviour policy, instead, is a policy that is used to collect the trajectory samples that will be used for learning. Depending on how the trajectories used to learn the target policy are collected, a learning algorithm can operate on-policy or off-policy:

- **On-policy**: the behaviour policy corresponds to the target policy. This means that every collected trajectory is sampled from the current policy. In simpler terms, it means that the agent learns from its own experience.
- **Off-policy**: the target policy differs from the behaviour policy. In this case, the agent will learn from 'someone else'. Note that 'someone else' can also mean from itself in a different time-frame.

Another difference between the learning algorithms arises when we consider the degree of knowledge about the environment that the agent has:

– **Model-based**: the agent has a model of the environment, which can be either exact or approximated. Note that if the agent has an exact model, then the MDP can be solved even without experience.

– **Model-free**: the agent has no information about the environment. The only way to learn is by collecting samples from the environment.

This concludes our partial taxonomy of RL methods. Of course, there are other aspects that are used to classify an RL method, but these are out of scope for this work. We will only consider model-free on-policy methods. This is a common scenario for control problems where it is possible to collect samples from the environment, but it may not be possible to build an exact model of it.
The following sections will describe the three main approaches that are used to solve an MDP: policy iteration, value iteration and policy search.

2.2.1 Policy iteration

As we have seen in the previous section, the state-value function associates a value to each state of an MDP by considering all the discounted future rewards that can be taken by following a policy π from that state. The state-value functions associated to different policies can be used to define an ordering relationship between policies:

Definition 2.11 (Policy improvement)**.** Given two policies π, π', policy π' is better than or equal to ($\succcurlyeq$) policy π when the value function of π' is greater than or equal to the value function of π in all states:

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

A policy $\pi' \succcurlyeq \pi$ can be found by taking, for each state, the action that yields the highest value. The policy so constructed is called greedy policy and can be found via maximisation over actions of the action-value function:

Definition 2.12 (Greedy policy)**.** The greedy policy is the policy that chooses the action that yields the highest value for each state:

$$\tilde{\pi}(a|s) = \arg\max_{a \in \mathcal{A}} Q^{\pi}(s, a), \qquad \forall s \in \mathcal{S}$$

The greedy policy $\tilde{\pi}$ is an improvement over the policy π as it occurs that $\tilde{\pi} \succcurlyeq \pi$. This result is known as the policy improvement theorem [7]. Note that the greedy policy $\tilde{\pi}$ of a policy π has an advantage function $A_{\pi}^{\tilde{\pi}}(s)$ that is non-negative in every state. Successive improvements require the evaluation of the newly found policy to obtain its associated action-value function.
Policy iteration method [8] aims at finding a sequence of policies $\pi^0 \succcurlyeq \pi^1 \succcurlyeq \pi^2 \ldots$ by alternating the improvement and the evaluation of the policy:

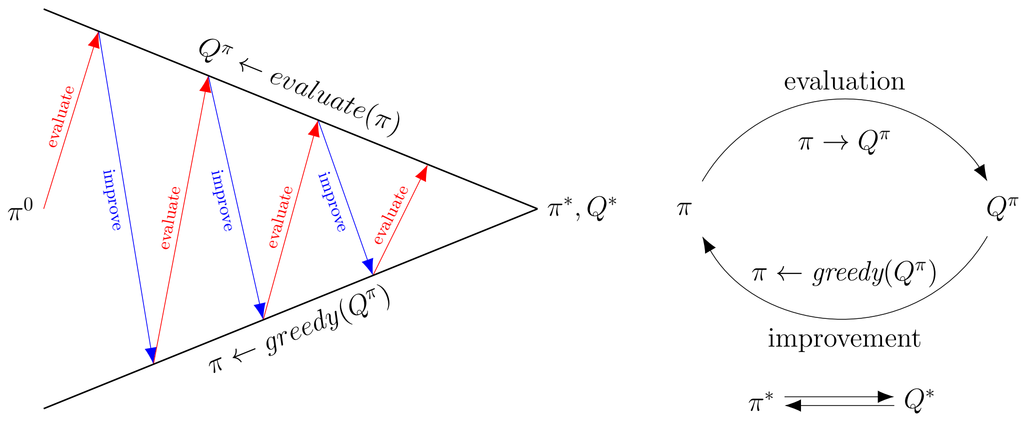


Figure 2.5: Visualization of the policy iteration algorithm. Starting from a policy π^0, we iteratively apply an evaluation and an improvement step, until we converge to the optimal action-value function $Q^*(s, a)$ and optimal policy π^*.

Algorithm 2.1 Policy iteration algorithm

```
Initialize $\pi^0$ arbitrarily
**for** t=0,1,2... until convergence **do**
    Evaluate policy $\pi^t$ and compute action-value function $Q^{\pi^t}(s, a)$
    Improve with greedy policy $\pi^{t+1}(a|s) \leftarrow \arg\max_a Q^{\pi^t}(s, a), \quad \forall s \in \mathcal{S}$
**end for**
```

- Policy evaluation: we evaluate the current policy π^k on the MDP and we compute the corresponding action-value function $Q^{\pi^k}(s, a)$.

- Policy improvement: we improve the current policy by computing the greedy policy from the computed action-value function: $\pi^{k+1} \leftarrow greedy\left(Q^{\pi^k}\right)$

This algorithm is guaranteed to converge to the optimal policy π^* when the state-action function $Q(s, a)$ can be computed exactly [9], [10], [11].
Figure 2.5 shows the steps of a policy iteration algorithm. A pseudocode of the algorithm is shown in Algorithm 2.1.
This approach, though, suffers from the following limitations:

- In practical scenarios, it is not possible to exactly evaluate the action-value function $Q^\pi(s, a)$, due to the stochasticity of the environment and the sampling nature of the algorithm.

- This method can only be applied to discrete MDPs, that is MDPs where both $\mathcal{S}$ and $\mathcal{A}$ are discrete sets. This is typically not the case for robotic control, where the state and action spaces are typically continuous.

Algorithm 2.2 Value iteration algorithm

```
Initialize $V^0$ arbitrarily
for t=0,1,2... until convergence do
    $V^{t+1}(s) = \max_a \left\{\mathcal{R}(s,a) + \gamma \int_{\mathcal{S}} \mathcal{P}(s'|s,a)V^t(s')\,\mathrm{d}s'\right\} \quad \forall s \in \mathcal{S}$
end for
Output policy $\pi^*(a|s) = \arg\max_a \left\{\mathcal{R}(s,a) + \gamma \int_{\mathcal{S}} \mathcal{P}(s'|s,a)V^t(s')\,\mathrm{d}s'\right\} \quad \forall s \in \mathcal{S}$
```

2.2.2 Value iteration

Value iteration algorithms try to overcome some of the problems of policy iteration algorithm, such as the possibly long time spent at evaluating the policy until convergence of the action-value function. The main idea of value iteration is to perform an improvement right after a 1-step evaluation of the policy. These two steps are merged together on a single update:

$$V^{t+1}(s) = \max_a \left\{\mathcal{R}(s,a) + \gamma \int_{\mathcal{S}} \mathcal{P}(s'|s,a)V^t(s')\,\mathrm{d}s'\right\} \quad \forall s \in \mathcal{S}$$

The above equation is known in literature as Bellman Optimality Equation, and is guaranteed to converge to the optimal state-value function $V^*(s)$ [7]. Value iteration effectively combines one sweep of policy evaluation and one sweep of policy improvement, removing any intermediate policy representation. Note that iterating solely on the value function may end up in unfeasible intermediate policies.
Algorithm 2.2 shows a pseudocode for a value iteration algorithm.

Model-free extensions to value iterations are based on temporal difference [12] and has proven to perform quite well. Among the most popular value-based algorithms we find Q-learning [13] and SARSA [14].

2.2.3 Policy search

The policy search technique is based on finding directly the optimal policy among a restricted set of policies Π, also called policy space. The search in this policy space can be carried out in different ways as a constrained optimization problem:

$$\text{find} \quad \pi^* = \arg\max_{\pi\in\Pi} J_\mu(\pi).$$

The most popular policy search method, which will be discussed in more details in Section 2.3, is policy gradient.
Policy gradient methods consider parameterized policies $\pi_{\boldsymbol{\theta}}$ and optimise such policies via gradient ascent on the policy parameter $\boldsymbol{\theta}$:

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

where α is called step-size or learning rate, and regulates the magnitude of the update. As we will see in Chapter 3, the choice of the step size α is extremely important for safe policy gradient approaches.

2.3 Policy gradient methods

So far we have considered mostly action-value methods, that is how to learn the action-value function $Q^\pi(s,a)$ and, from this function, obtain the target policy. Policy iteration methods switch between policy evaluation and policy improvement using the greedy policy $\tilde{\pi}$, while value iteration methods aim at finding directly the optimal action-value function $Q^*(s,a)$ with an iterative approach. Policy gradient, instead, is a policy search method that does not consider the action-value function, but considers a set of parameterized policies $\pi_{\boldsymbol{\theta}}(a|s) = \pi(a|s,\boldsymbol{\theta})$, with $\boldsymbol{\theta} \in \mathbb{R}^m$. We define the policy space Π as the set of all possible parameterized policies $\Pi : \{\pi_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^m\}$. The aim of policy gradient methods is to find an optimal parameter vector $\boldsymbol{\theta}^* \in \Theta$ such that $J_\mu(\pi_{\boldsymbol{\theta}})$ is maximum. In this perspective, the value function is no longer required, though it could be exploited to learn $\boldsymbol{\theta}^*$.

Common parametrizations are the soft-max (for discrete action space) and the Gaussian (for continuous action space):

Soft-max: $$\pi_{\boldsymbol{\theta}}(a|s) = \frac{\exp\left(\boldsymbol{\theta}^\mathsf{T}\phi(s,a)\right)}{\sum_{a'\in\mathcal{A}}\exp\left(\boldsymbol{\theta}^\mathsf{T}\phi(s,a')\right)}.$$

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

where the mean $\mu_{\boldsymbol{\theta}} = \boldsymbol{\theta}^\mathsf{T}\phi(\cdot)$ can be replaced by any differentiable approximation function. In this work, we will only consider linear-mean Gaussian-parametrized policies, which are very common in practice.

The performance of a parametrized policy can be maximised by performing gradient ascent on its parameter:
$$\boldsymbol{\theta}^{t+1} \leftarrow \boldsymbol{\theta}^t + \alpha\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}),$$

where $\alpha \geq 0$ denotes the learning rate, also called step size. Gradient descent methods guarantee convergence at least to a local optimum. The quantity $\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$ is called policy gradient (hence the name of the method) and can be computed analytically:

Theorem 2.1 (Policy gradient theorem from [15])**.** *For any MDP:*

$$\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) = \int_\mathcal{S} d_\mu^{\pi_{\boldsymbol{\theta}}}(s)\int_\mathcal{A}\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a|s)Q^{\pi_{\boldsymbol{\theta}}}(s,a)\,da\,ds$$

The policy gradient theorem provides an analytic expression for the gradient of the performance with respect to the policy parameters. Note that this expression does not involve the derivative of the state distribution, whose quantity may be very hard to estimate.
In many tasks it is not possible to compute exactly the quantity $\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$, since the future state distribution $d_\mu^{\pi_{\boldsymbol{\theta}}}(s)$ and the action-value function $Q^{\pi_{\boldsymbol{\theta}}}(s,a)$ are not

known. For this reason, the gradient w.r.t. the policy performance needs to be estimated from samples. Finding a good estimate of $\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$ constitutes one of the main challenges for policy gradient methods. The next section will provide two methods for gradient estimation called finite difference and likelihood ratio.

2.3.1 Gradient estimation

The main problem in policy gradient methods is to obtain a good estimator of the policy gradient $\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$. Traditionally, researchers have used model-based methods for obtaining the gradient [16], [17]. However, in order to operate in model-free scenario, we cannot employ a model of the environment and the estimation must rely only on samples drawn from our experience. The gradient estimate is often averaged over N trajectories, where N is usually called batch size. For this reason, we will perform Stochastic Gradient Descent (SGD) on the policy parameters, that is still guaranteed to converge at least to a local optimum [18]. An implementation of a policy gradient method that employs SGD is shown in Algorithm 2.3.
The literature on policy gradient has yielded a variety of methods to estimate $\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$, but here we will present the two most prominent approaches: finite difference and likelihood ratio.

Finite difference Finite difference methods aim at finding the policy gradient by adding small variations $\Delta\boldsymbol{\theta}$ to the policy parameters. For each increment $\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_i$ several roll-outs are performed to calculate the expected return difference $\Delta\hat{J}_i \simeq J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}_i) - J(\boldsymbol{\theta})$.
The policy gradient estimate is then calculated by solving a regression problem:

$$\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) = \left(\Delta\boldsymbol{\Theta}^\mathsf{T}\Delta\boldsymbol{\Theta}\right)^{-1} \Delta\boldsymbol{\Theta}^\mathsf{T}\Delta\hat{\boldsymbol{J}},$$

where $\Delta\boldsymbol{\Theta}$ contains the perturbations $\Delta\boldsymbol{\theta}_i$ and $\Delta\hat{\boldsymbol{J}}$ contains the return differences $\Delta\hat{J}_i$.

This approach is general, as it does not require any specific knowledge about the system, and it is straightforward to apply, however in real systems it suffers from slow convergence and requires a huge number of sample trajectories.

Likelihood ratio Likelihood ratio methods [19], later refined with the REINFORCE trick [20], use a different approach to estimate the policy gradient.
The REINFORCE trick employs this general identity:

$$\nabla \log f = \frac{\nabla f}{f}$$

to the policy gradient: $\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) = \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)$ so that we can estimate the future state distribution from samples taken from $\pi_{\boldsymbol{\theta}}$. We can rewrite the policy gradient in this way:

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

that can be estimated via roll-outs of the policy $\pi_{\boldsymbol{\theta}}$:

$$\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}, s_0\sim\mu} \left[\nabla_{\boldsymbol{\theta}} \log \pi(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s,a)\right].$$

The estimation of the action-value function can be done with any of the methods seen in the previous section, but a straightforward way to estimate it is directly from discounted rewards: $\hat{Q}^{\pi_{\boldsymbol{\theta}}}(s,a) \simeq \sum_{k=0}^{H} \gamma^k \mathcal{R}(s_k, a_k)$. Using this latter approximation, we obtain the REINFORCE estimator as proposed by Williams [20]:

$$\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})_{RF} = \left\langle \left(\sum_{k=0}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k) \right) \left(\sum_{k=0}^{H} \gamma^k \mathcal{R}(s_k, a_k) \right) \right\rangle_N \tag{2.1}$$

where N is the batch size and $\langle\cdot\rangle_N$ denotes the sample mean over N trajectories. The advantage of this approach with respect to finite differences is that it is possible to have a gradient estimation even from a single roll-out and it is guaranteed to converge. Besides these factors, REINFORCE gradient estimator suffers from high variance, that grows at least cubically with the length of the horizon and quadratically with the magnitude of the reward.
A better approach comes from the observation that future actions do not depend on past rewards, leading to two formulations from [15]:

$$\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})_{PGT} = \left\langle \sum_{k=0}^{H} \gamma^k \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k) \left(\sum_{l=k}^{H} \gamma^{l-k} \mathcal{R}(s_l, a_l) \right) \right\rangle_N$$

known as PGT and from [21], known as G(PO)MDP:

$$\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})_{GMDP} = \left\langle \sum_{l=0}^{H} \left(\sum_{k=0}^{l} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k) \right) (\gamma^l \mathcal{R}(s_l, a_l) \right\rangle_N$$

that were lately proven to be exactly equivalent as a result of the summation theorem [22], that is: $\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})_{PGT} = \hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})_{GMDP}$. This estimator was found to have lower variance than the REINFORCE estimator [21].

2.3.2 Baselines

Another approach that was used to reduce the variance of the REINFORCE and G(PO)MDP estimators was to add an action-independent value $b(s)$, also called baseline, resulting in an unbiased estimation of the gradient. In fact, the following result applies:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) &= \mathbb{E}_{\pi,\mu} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q^{\pi_{\boldsymbol{\theta}}}(s,a)\right] \\ &= \mathbb{E}_{\pi,\mu} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \left(Q^{\pi_{\boldsymbol{\theta}}}(s,a) - b(s)\right)\right]\end{aligned}$$

Algorithm 2.3 REINFORCE algorithm with baseline

```
Initialize $\boldsymbol{\theta} \in \Theta$ randomly
for t=0,1,2... until convergence do
    Collect $N$ trajectories $s_0, a_0, r_0, \dots s_H, a_H, r_H$.
    Compute baseline $b = \frac{\left\langle \left(\sum_{k=0}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k)\right)^2 \sum_{l=0}^{H} \gamma^l \mathcal{R}(s_l,a_l) \right\rangle}{\left\langle \left(\sum_{k=0}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k)\right)^2 \right\rangle}$
    Estimate $\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) = \left\langle \left(\sum_{k=0}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k)\right) \left(\sum_{k=0}^{H} \gamma^k \mathcal{R}(s_k, a_k) - b\right) \right\rangle_N$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$
end for
```

for any action-independent value $b \in \mathbb{R}$ [20]. Exploiting this result, we can find the baseline b such that the variance of the estimator is minimized, yielding the following results for REINFORCE and G(PO)MDP as shown in [1]:

$$\textbf{REINFORCE:} \qquad b = \frac{\left\langle \left(\sum_{k=0}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k)\right)^2 \sum_{l=0}^{H} \gamma^l \mathcal{R}(s_l, a_l) \right\rangle}{\left\langle \left(\sum_{k=0}^{H} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k)\right)^2 \right\rangle},$$

$$\textbf{G(PO)MDP:} \qquad b_{\mathrm{k}} = \frac{\left\langle \left(\sum_{k=0}^{\mathrm{k}} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k)\right)^2 \gamma^{\mathrm{k}} \mathcal{R}(s_{\mathrm{k}}, a_{\mathrm{k}}) \right\rangle}{\left\langle \left(\sum_{k=0}^{\mathrm{k}} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_k|s_k)\right)^2 \right\rangle}.$$

2.3.3 Natural Policy Gradient

Despite all the advances made so far to have good estimates of the performance gradient w.r.t. the policy parameters, the policy gradient methods seen above may still perform poorly in some situations [1]. One of the reasons is that the computed gradients strongly depend on the policy parametrization. Natural gradient methods aim to compute gradients that are independent from the policy parametrization. The main observation behind natural gradient is that, when we perform an update $\Delta\boldsymbol{\theta}$ on the policy parameters and we use Euclidean metric of $\sqrt{\boldsymbol{\theta}^\mathsf{T}\boldsymbol{\theta}}$, the gradient is different for every parametrization $\boldsymbol{\theta}$ of the policy $\pi_{\boldsymbol{\theta}}$. Natural gradient [23] overcomes this problem by applying a linear transformation to the gradient [1]. This linear transformation yields the following parameter update:

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

where:

$$\begin{aligned} F_{\boldsymbol{\theta}} &= \int_{\mathcal{S}} d_\mu^{\pi_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a, s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)^\mathsf{T} \,\mathrm{d}a\,\mathrm{d}s \\ &\simeq \left\langle \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)^\mathsf{T} \right\rangle \end{aligned}$$

is the Fisher information matrix.
This method is guaranteed to converge [24] as the effective gradient is rotated by an angle that is less than $\pi/2$. Natural Policy Gradient has been shown to be effective in practice [25].

Chapter 3

Safe Policy Improvement

In this chapter, we will describe some problems that can arise from RL techniques and why it is important to develop safe learning algorithms.
We will start with the motivation of safe policy improvement in Section 3.1, then dive into the details of several methods to address this problem. We will start from Conservative Policy Iteration (CPI) [2], in Section 3.2, that is considered one of the seminal works in this field. This work, in fact, sets the basis for several further developments. Pirotta et al. [3] studied a more general bound for policy iteration, described in Section 3.3, that was later applied to policy gradient methods [4] (Section 3.4). On this line of research, Papini et al. [5] considered also an adaptive batch size, as explained in Section 3.5. Following a different approach, Shulman et al. [26] extended the result from CPI and developed a general algorithm called Trust Region Policy Optimization (TRPO), as described in Section 3.6. Finally, in Section 3.7, we will briefly describe other approaches to safe policy improvements that no longer require monotonic improvement.

3.1 Safety in Reinforcement Learning

In this section, we will formalize the concept of safety in reinforcement learning. While the learning methods presented in Chapter 2 can perform well in the exact case, some problems arise when we consider model-free control. In this latter case, in fact, we have to estimate some quantities and the uncertainty of these estimates can harm the online performance.
More specifically the two main sources of approximations are:

– The length of the collected trajectories is limited when we consider an infinite-horizon MDP. Hence, the cumulative reward function becomes a truncated sum, thus introducing an approximation error. This truncation can add a bias term in the estimations.

- The number of trajectories used to estimate the value functions or the policy gradient is limited. The number of collected trajectories is usually called batch size. This factor introduces variance in the estimations. Using large batch sizes can limit the problem (reduce variance), but this comes to the cost of collecting more samples, which may be unfeasible in many applications.

These approximation errors induce some oscillations in the policy performance, either because the update direction is wrong or the magnitude of the estimate is too high and we incur in an overshooting. Even though these oscillations can be tolerable during the learning phase in some applications, it may become unbearable in some other situations that require a more reliable approach (e.g. robotic control). This is known in literature as the Policy-Oscillation Problem [27], [28]. These oscillations can affect learning in several ways:

- **Dangerous behaviours**: in an on-line scenario, a low peak in policy performance can result in a critical interaction with the environment. This can be the case of Autonomous Driving, where a bad action choice can cause several problems.
- **Poor on-line performance**: consider an automatic production line where we want to improve the total production over several days while maintaining the production high every day. A low peak on performance negatively impacts production and generates costs.
- Selection of **sub-optimal policies**: if we stop the learning phase after a fixed number of iterations, we may end up in a low peak. Detecting the low peak and manually reject previous policies can be a difficult and sometimes prohibitive task.

Safe approaches to reinforcement learning aims at guaranteeing performance improvements in spite of approximations, while maintaining the costs of collecting samples to an acceptable level. In order to start with the several approaches that tries to overcome these problems, we summarize the above statements and formalize the concept of a safe learning algorithm:

Definition 3.1 (Safe Learning Algorithm)**.** Given a (user-specified) lower-bound $\underline{J}$ on the performance and a confidence level δ, we call an RL algorithm safe if it ensures that the probability that a policy with performance less than $\underline{J}$ will be proposed is at most δ.

Note that this is a general definition and accounts for safe improvements in high probability. A special case can be defined when the reference bound $\underline{J}$ is set to the performance of the policy at the previous iteration: $\underline{J}^{(t)} = J^{(t-1)}$. In this case we can guarantee monotonic improvement.
The proposed methods below will consider this latter monotonic improvement case, with the exception of Section 3.7, that will consider the more general case where $\underline{J}$ is user-defined.

3.2 Conservative Policy Iteration

Kakade and Langford [2] proposed a novel solution to guarantee monotonic improvement in a policy iteration method. They faced the policy oscillation problem by limiting the magnitude of the policy updates in the following form:

$$\pi' = (1-\alpha)\pi + \alpha\overline{\pi}, \tag{3.1}$$

where the current policy π is updated with a convex combination between the current policy π and the greedy policy $\overline{\pi}$ with a factor $\alpha \in [0,1]$, also called learning factor. When $\alpha = 1$ we have the usual policy iteration as described in Chapter 2. This update is not guaranteed to be safe in general. Kakade considered a conservative approach to safety by choosing $\alpha < 1$ and making smaller updates that guarantee an improvement. A first result achieved by Kakade is summarised in Theorem 3.1:

Theorem 3.1 (Adapted from Theorem 4.1, Kakade 2002)**.** *Assuming a policy update of the form 3.1 in a policy iteration setting, the following performance improvement is guaranteed for all $\alpha \in [0,1]$:*

$$J_\mu(\pi') - J_\mu(\pi) \geq \frac{\alpha}{1-\gamma}\left(\mathbb{A}_{\pi,\mu}^{\overline{\pi}} - \frac{2\alpha\gamma\epsilon}{1-\gamma(1-\alpha)}\right), \tag{3.2}$$

where:

$$\mathbb{A}_{\pi,\mu}^{\overline{\pi}} = \mathbb{E}_{s\sim d_{\pi,\mu}}\left[\mathbb{E}_{a\sim\overline{\pi}}\left[A_\pi(s,a)\right]\right] \tag{3.3}$$

is the policy advantage function and

$$\epsilon = \max_s \left|\mathbb{E}_{a\sim\overline{\pi}}\left[A_\pi(s,a)\right]\right|. \tag{3.4}$$

This result can be analysed as follows: the first term is analogous to the first order expansion of the performance measure J with respect to α and is related to the policy advantage function, while the second term is a penalization term that prevents taking large values of α. These two terms are in contrast, thus posing a trade-off condition between the expected advantage given by the first term and the magnitude of the update, that can harm monotonic improvements, given by the second term. Given this result, an algorithm can be developed as follows:

Theorem 3.2 (Adapted from Corollary 4.2, Kakade 2002)**.** *Let R be the maximum possible reward. Using update rule 3.1, if $\mathbb{A}_{\pi,\mu}^{\overline{\pi}} \geq 0$ the performance bound 3.2 is maximised by choosing:*

$$\alpha^* = \frac{(1-\gamma)\mathbb{A}_{\pi,\mu}^{\overline{\pi}}}{4R}, \tag{3.5}$$

guaranteeing a policy improvement of:

$$J_\mu(\pi') - J_\mu(\pi) \geq \frac{{\mathbb{A}_{\pi,\mu}^{\overline{\pi}}}^2}{8R}. \tag{3.6}$$

Algorithm 3.1 Conservative Policy Iteration

```
Initialize policy $\pi^0$ arbitrarily.
for $t = 0, 1, 2 \dots$ until convergence do
    Evaluate current policy $\pi^t$
    Compute target $\overline{\pi}(s) \in \arg\max_{a \in \mathcal{A}} \hat{Q}^{\pi}(s, a)$
    Compute advantage policy function $\mathbb{A}_{\pi,\mu}^{\overline{\pi}}$
    Compute $\alpha^* = \min \left\{ \frac{(1-\gamma)\mathbb{A}_{\pi,\mu}^{\overline{\pi}}}{4R}, 1 \right\}$
    Update policy $\pi^{t+1} \leftarrow \alpha^* \pi^t + (1 - \alpha^*)\overline{\pi}^t$
end for
```

This algorithm is a generalization of the policy iteration algorithm introduced in Chapter 2 as it adapts the magnitude of the update with a learning factor α in order to maximize the guaranteed improvement $J_\mu(\pi') - J_\mu(\pi)$. This algorithm was named Conservative Policy Iteration (CPI) after its tendency to make conservative policy updates. The pseudo-code of CPI is reported in Algorithm 3.1.

It is also useful to report a notable and general result from Kakade, 2002 that will be used as a starting point for more advanced bounds in the next sections:

Theorem 3.3 (Lemma 6.1 from Kakade 2002). *For any policies $\tilde{\pi}$ and π and any starting state distribution μ,*

$$J_\mu(\tilde{\pi}) - J_\mu(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{(a,s) \sim \tilde{\pi}, d_{\tilde{\pi},\mu}} [A_\pi(s, a)]. \tag{3.7}$$

3.3 Safe policy iteration

The approach used by Kakade and Langford was really smart and influenced many later works in the last decade. The contribution, though, was mainly theoretical, since the proposed algorithm (CPI) was found to be over-conservative, leading to a slow learning process due to the looseness of the bound.
This problem was addressed by Pirotta et al. [3] that derived a tighter bound on policy improvement w.r.t. the one in CPI. They also proposed new algorithms that exploit this tighter bound and provided experimental results.

Before diving into the derivation of the bound on performance improvement, Pirotta et al. derived some bounds on the future state distribution d_μ^π, that are reported here for their generality:

Theorem 3.4 (Adapted from Corollary 3.2 of [3]). *Let π and π' be two stationary policies for an infinite horizon MDP $\mathcal{M}$. The L_1-norm of the difference between their γ-discounted future state distributions under starting state distribution μ can be upper bounded as follows:*

$$\left\| d_\mu^{\pi'} - d_\mu^{\pi} \right\|_1 \leq \frac{\gamma}{(1-\gamma)^2} D_\infty^{\pi,\pi'}, \tag{3.8}$$

where:

$$D_{\infty}^{\pi,\pi'} = \sup_{s\in\mathcal{S}} \left\|\pi'(\cdot \mid s) - \pi(\cdot \mid s)\right\|_1 . \tag{3.9}$$

This is a (looser) model-free version of the bound from [3], where the difference between the two distributions depends only on the discount factor γ and the difference between the two policies.
The authors then used Theorem 3.3 to develop a tighter bound on performance improvement:

Theorem 3.5. *For any stationary policies π and π' and any starting state distribution μ, given any baseline policy π_b, the difference between the performance of π' and the one of π can be lower bounded as follows*[1]*:*

$$J_\mu(\pi') - J_\mu(\pi) \geq \frac{d_\mu^{\pi_b \intercal} A_\pi^{\pi'}}{1-\gamma} - \frac{\gamma}{(1-\gamma)^2} D_\infty^{\pi_b,\pi'} \frac{\Delta A_\pi^{\pi'}}{2}, \tag{3.10}$$

where:

$$\Delta A_\pi^{\pi'} = \sup_{s,s'\in\mathcal{S}} \left| A_\pi^{\pi'}(s) - A_\pi^{\pi'}(s') \right| . \tag{3.11}$$

This bound is similar to Equation (3.2). The main difference is in the penalty term that here depends on two factors: the variation in the advantage function $\Delta A_\pi^{\pi'}$ and the distance between the two policies π and π' in infinite norm. Since approximating the advantage function may be difficult in some problems, the authors also provided a looser but simplified bound that requires the action-value function instead:

Theorem 3.6. *For any stationary policies π and π' and any starting state distribution μ, the difference between the performance of π' and the one of π can be lower-bounded as follows:*

$$J_\mu(\pi') - J_\mu(\pi) \geq \frac{\mathbb{A}_{\pi,\mu}^{\pi'}}{1-\gamma} - \frac{\gamma}{(1-\gamma)^2} D_\infty^{\pi,\pi'^2} \frac{\|Q^\pi\|_\infty}{2}, \tag{3.12}$$

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

Pirotta et al. provided two algorithms that employ this bound for safe policy iteration. The first proposed algorithm is called Unique-parameter Safe Policy Improvement (USPI), that uses the update rule defined in Equation (3.1) with $\alpha \in [0,1]$ to update the current policy. In this setting, the following theorem holds:

Theorem 3.7. *If $\mathbb{A}_{\pi,\mu}^{\overline{\pi}} \geq 0$ then, using update rule 3.2 with $\alpha = \min(\alpha^*, 1)$, with $\alpha^* = \frac{(1-\gamma)\mathbb{A}_{\pi,\mu}^{\overline{\pi}}}{\gamma D_\infty^{\pi,\pi'} \Delta A_\pi^{\pi'}}$ we have:*

[1] In the original paper, Pirotta et al. uses the un-normalized state occupancy $d_\mu^\pi = \sum_{t=0}^\infty \gamma^t Pr(s_t = s \mid \pi, \mu)$ instead of the normalized future state distribution $d_\mu^\pi = (1-\gamma)\sum_{t=0}^\infty \gamma^t Pr(s_t = s \mid \pi, \mu)$ as in this work. This introduces a normalization factor $1-\gamma$ that has to be considered.

– $\alpha^* \leq 1$*: the following policy improvement is guaranteed:*

$$J_\mu(\pi') - J_\mu(\pi) \geq \frac{{\mathbb{A}_{\pi,\mu}^{\pi'}}^2}{2\gamma D_\infty^{\pi,\overline{\pi}} \Delta A_\pi^{\overline{\pi}}}.$$

– $\alpha^* > 1$*: we have a full update toward* $\overline{\pi}$ *and the guarantee from Theorem 3.5 setting* $\pi_b = \pi$*:*

$$J_\mu(\pi') - J_\mu(\pi) \geq \frac{\mathbb{A}_{\pi,\mu}^{\overline{\pi}}}{1-\gamma} - \frac{\gamma}{(1-\gamma)^2} D_\infty^{\pi,\pi'} \frac{\Delta A_\pi^{\pi'}}{2}. \tag{3.13}$$

The underlying algorithm for USPI is similar to Algorithm 3.1, with the additional evaluation of $D_\infty^{\pi,\overline{\pi}}$ and $\Delta A_\pi^{\pi'}$, and the choice of the step size α^* as in Theorem 3.7.

The second algorithm proposed by Pirotta et al. is called Multiple-parameter Safe Policy Improvement (MSPI) and employs a policy update rule of the following form:

$$\pi'(a \mid s) = \alpha(s)\overline{\pi}(a \mid s) + (1 - \alpha(s))\pi(a \mid s), \qquad \forall s, a,$$

where $\alpha(s) \in [0, 1] \forall s$ is a per-state convex-combination coefficient. This more general variant proved to be only slightly better than USPI in some domain but it often happened that the policy improvement bound found by MSPI was lower than the one of USPI, because the former, due to its increased complexity, optimises the bound in Theorem 3.6, while USPI can optimise the tighter bound in Theorem 3.5

3.4 Adaptive Step Size For PG

Pirotta et al. [4] further developed their work on safe policy iteration and adapted the bound from [3] and described in Section 3.3 to be used with policy gradient methods.
Their first contribution is a continuous version of the bound described in Theorem 3.6:

Theorem 3.8 (Continuous MDP version of Corollary 3.6 in [3]). *For any pair of stationary policies corresponding to parameters* $\boldsymbol{\theta}$ *and* $\boldsymbol{\theta}'$ *and for any starting state distribution* μ*, the difference between the performance of policy* $\pi_{\boldsymbol{\theta}'}$ *and* $\pi_{\boldsymbol{\theta}}$ *can be bounded as follows:*

$$J_\mu(\boldsymbol{\theta}') - J_\mu(\boldsymbol{\theta}) \geq \frac{1}{1-\gamma} \int_{\mathcal{S}} d_\mu^{\pi_{\boldsymbol{\theta}}}(s) A_{\pi_{\boldsymbol{\theta}}}^{\pi_{\boldsymbol{\theta}'}}(s)\, ds - \frac{\gamma}{2(1-\gamma)^2} \left\| \pi_{\boldsymbol{\theta}'} - \pi_{\boldsymbol{\theta}} \right\|_\infty^2 \left\| Q^{\pi_{\boldsymbol{\theta}}} \right\|_\infty,$$

where $\left\| Q^{\pi_{\boldsymbol{\theta}}} \right\|_\infty$ *is the supremum norm of the Q-function:* $\left\| Q^{\pi_{\boldsymbol{\theta}}} \right\|_\infty = \sup_{s\in\mathcal{S}, a\in\mathcal{A}} Q^{\pi_{\boldsymbol{\theta}}}(s, a)$.

Here again we have two terms, the first related with the policy advantage function $\mathbb{A}_{\pi_{\boldsymbol{\theta}},\mu}^{\pi_{\boldsymbol{\theta}'}}$, that encodes how much the target policy $\pi_{\boldsymbol{\theta}'}$ performs better than $\pi_{\boldsymbol{\theta}}$, while

the second term is a penalty term driven by the distance between the two policies in infinite norm.
Following this extension, they provided a new formulation for the difference between two policies that was used to build the following bound for the update rule $\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$:

Theorem 3.9 (Lemma 3.2 from [4]). *Let the update of the policy parameters be $\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$. Then for any stationary policy $\pi(a|s,\boldsymbol{\theta})$ and any starting state distribution μ, the difference between $\pi_{\boldsymbol{\theta}}$ and $\pi_{\boldsymbol{\theta}'}$ is lower bounded by:*

$$\begin{aligned}
&J_\mu(\boldsymbol{\theta}') - J_\mu(\boldsymbol{\theta}) \geq \alpha \left\|\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})\right\|_2^2 \\
&+ \frac{\alpha^2}{1-\gamma} \int_{\mathcal{S}} d_\mu^{\pi_{\boldsymbol{\theta}}}(s) \int_{\mathcal{A}} \inf_{c \in (0,1)} \left(\sum_{i,j=1}^{m} \left. \frac{\partial^2 \pi(a|s,\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j} \right|_{\boldsymbol{\theta}+c\Delta\boldsymbol{\theta}} \frac{\Delta\theta_i \Delta\theta_j}{1+I(i=j)} \right) Q^{\pi_{\boldsymbol{\theta}}}(s,a) da ds \\
&- \frac{\gamma \left\|Q^{\pi_{\boldsymbol{\theta}}}\right\|_\infty}{2(1-\gamma)^2} \left(\alpha \sup_{s\in\mathcal{S}} \int_{\mathcal{A}} \left| \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})^\mathsf{T} \nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) \right| da \right. \\
&\qquad \left. + \alpha^2 \sup_{s\in\mathcal{S}} \int_{\mathcal{A}} \left| \sup_{c\in(0,1)} \left(\sum_{i,j=1}^{m} \left. \frac{\partial^2 \pi(a|s,\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j} \right|_{\boldsymbol{\theta}+c\Delta\boldsymbol{\theta}} \frac{\Delta\theta_i \Delta\theta_j}{1+I(i=j)} \right) \right| da \right)^2,
\end{aligned}$$

where $\Delta\boldsymbol{\theta} = \alpha \nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$.

The bound above is a fourth order polynomial of the step size, whose stationary points can be expressed in closed form. Using Descartes' rule we can prove the existence and uniqueness of a real positive value for the optimal step size α that maximises the guaranteed performance improvement.
This complex bound, although very general, can be simplified by adding an assumption on the policy class. Since we are considering continuous MDPs, it is reasonable to employ Gaussian policies of the following form:

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

where the mean is a linear combination of a parameter vector $\boldsymbol{\theta}$ of size m and a state feature vector $\phi(\cdot)$.
Moreover, the authors assumed that features ϕ are uniformly bounded:

$$|\phi_i(s)| < M_\phi, \qquad \forall s \in \mathcal{S}, \forall i = 1, \ldots, m. \tag{3.14}$$

With these assumptions, the above bound in Theorem 3.9 becomes a more handy second-order polynomial that can be easily employed in a practical algorithm:

Theorem 3.10 (Corollary 5.1 from [4]). *For any starting state distribution μ, and any pair of stationary Gaussian policies $\pi_{\boldsymbol{\theta}} \sim \mathcal{N}(\boldsymbol{\theta}^\mathsf{T}\phi(s), \sigma^2)$ and $\pi_{\boldsymbol{\theta}'} \sim \mathcal{N}(\boldsymbol{\theta}'^\mathsf{T}\phi(s), \sigma^2)$, so that $\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$, under assumption 3.14, the difference between the performance of $\pi_{\boldsymbol{\theta}'}$ and the one of $\pi_{\boldsymbol{\theta}}$ can be lower bounded as follows:*

$$J_\mu(\boldsymbol{\theta}') - J_\mu(\boldsymbol{\theta}) \geq \alpha \left\|\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})\right\|_2^2 - \alpha^2 \frac{R M_\phi^2 \left\|\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})\right\|_1^2}{(1-\gamma)^2\sigma^2} \left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)} \right),$$

where R is the maximum absolute reward and $|\mathcal{A}|$ is the volume of the action space.

This simplified bound has a single maximum attained for some positive value of α. The following statement can be used to design a policy gradient algorithm with adaptive step size:

Corollary 3.11. *The performance lower bound provided in Theorem 3.10 is maximised by choosing the following step size:*

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

that guarantees the following policy performance improvement

$$J_\mu(\boldsymbol{\theta}') - J_\mu(\boldsymbol{\theta}) \geq \frac{1}{2}\alpha^* \left\|\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})\right\|_2^2.$$

3.4.1 Approximate framework

The simplified bound in Theorem 3.10 depends on some constants and on the policy gradient $\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$. In a model-free control problem, we cannot compute the policy gradient directly, hence it has to be estimated from experience samples. This approximation of the policy gradient introduces some noise that can potentially harm the safety of the algorithm.
To address this issue, Pirotta et al. [4] considered the policy gradient to be a random variable $\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$ that has a bounded error ϵ with respect to the true value $\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$ so that, with probability at least $1-\delta$ we have:

$$\mathbb{P}\left(\left|\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) - \hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})\right| \geq \epsilon\right) \leq \delta.$$

Given this formalization of the policy gradient estimate, we can guarantee a performance improvement bound in high probability:

Theorem 3.12 (Theorem 5.2 from [4]). *Under the same assumptions of Theorem 3.10 and provided that a gradient estimate $\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$ is available so that $\mathbb{P}\left(\left|\nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) - \hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})\right| \geq \epsilon\right) \leq \delta$, the difference between the performance of $\pi_{\boldsymbol{\theta}'}$ and the one of $\pi_{\boldsymbol{\theta}}$ can be lower bounded with probability at least $(1-\delta)^m$ as follows:*

$$J_\mu(\boldsymbol{\theta}') - J_\mu(\boldsymbol{\theta}) \geq \alpha \left\|\underline{\hat{\nabla} J_\mu(\boldsymbol{\theta})}\right\|_2^2 - \alpha^2 \frac{RM_\phi^2 \left\|\overline{\hat{\nabla} J_\mu(\boldsymbol{\theta})}\right\|_1^2}{(1-\gamma)^2\sigma^2}\left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)}\right),$$

where $\underline{\hat{\nabla} J_\mu(\boldsymbol{\theta})} = \max(|\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})| - \epsilon, 0)$ and $\overline{\hat{\nabla} J_\mu(\boldsymbol{\theta})} = |\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})| + \epsilon$.
This bound is maximised by the following step size:

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

The above bound holds in high probability when the estimate of the policy gradient $\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$ is computed from a number of trajectories that is large enough. The batch

size that is required to guarantee such constraint depends on the method used to estimate the gradient and the concentration bound used, to model ϵ.
The authors used the Chebyshev's inequality to provide a high-probability upper bound to the gradient approximation error for the REINFORCE estimator and for the G(PO)MDP/PGT estimator, leading to the following result:

Corollary 3.13 (Adapted from Theorem 5.4 and 5.6 from [4])**.** *The number of H-step trajectories required to guarantee, with probability at least* $(1-\delta)$, *that*

$$\left|\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})\right| \leq \epsilon$$

is:

- *REINFORCE:* $N = \frac{R^2 M_\phi^2 H (\gamma^H)^2}{\delta \epsilon^2 \sigma^2 (1-\delta)^2}$,
- *G(PO)MDP/PGT:* $N = \frac{R^2 M_\phi^2}{\delta \epsilon^2 \sigma^2 (1-\gamma)^2} \left[\frac{1-\gamma^{2H}}{1-\gamma^2} + H\gamma^{2H} - 2\gamma^H \frac{1-\gamma^H}{1-\gamma}\right]$.

3.5 Adaptive Batch Size

Pirotta et al. [4] showed that a safe policy gradient algorithm can be developed by carefully choosing the step size α that maximizes a bound on performance improvement. However, in the approximate framework we have seen that the policy gradient has to be estimated and is thus subject to possibly high variance. This problem weighs on the choice of the batch size N. Using some general (but very loose) concentration bounds, such as Chebyshev's inequality, Pirotta et al [4]. provided an estimate for the batch size to use. However, the magnitude of this estimate is too conservative as it would require a number of trajectory samples that is prohibitive in most cases.
Papini et al. [5] addressed this problem by adapting the number of samples required to guarantee a high probability performance improvement based on the current statistics of the policy gradient estimate $\hat{\nabla}_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$. This approach is very innovative since the batch size, in the literature, is usually hand-tuned to obtain the best performance in each domain. Their first contribution is an extension of the bound in Theorem 3.10 for the general case where the step size is non-scalar: $\boldsymbol{\theta}' = \boldsymbol{\theta} + \Lambda \nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$, $\Lambda = diag(\alpha_1, \alpha_2, \dots, \alpha_m)$. Under the same assumption stated in Equation (3.14), they produced the following result:

Theorem 3.14. *For any initial state distribution* μ *and any pair of stationary Gaussian policies* $\pi_{\boldsymbol{\theta}} \sim \mathcal{N}(\boldsymbol{\theta}^\mathsf{T} \phi(s), \sigma^2)$ *and* $\pi_{\boldsymbol{\theta}'} \sim \mathcal{N}(\boldsymbol{\theta}'^\mathsf{T} \phi(s), \sigma^2)$ *so that* $\boldsymbol{\theta}' = \boldsymbol{\theta} + \Lambda \nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})$, *the difference between the performance of* $\pi_{\boldsymbol{\theta}'}$ *and the one of* $\pi_{\boldsymbol{\theta}}$ *can be lower bounded as follows:*

$$J_\mu(\boldsymbol{\theta}') - J_\mu(\boldsymbol{\theta}) \geq \nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})^\mathsf{T} \Lambda \nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta}) - c \left\|\Lambda \nabla_{\boldsymbol{\theta}} J_\mu(\boldsymbol{\theta})\right\|_1^2,$$

where $c = \frac{RM_\phi^2}{(1-\gamma)^2\sigma^2}\left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)}\right)$.
This lower bound is maximised by the following non-scalar step size:

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

Upon this result, Papini et al. introduced a cost-sensitive performance improvement measure $\Upsilon_\delta(\Lambda, N) = \frac{B_\delta(\Lambda,N)}{N}$ that relates the high-probability lower bound on performance improvement $B_\delta(\Lambda, N)$ with the batch size N, as done in [29]. The idea is to maximise the performance improvement per sample trajectory. This poses an interesting trade-off, since larger batch sizes lead to more accurate policy updates but this gain is spread across the multiple trajectories, while smaller batch sizes can also worsen the performance due to the high variance of the policy gradient.
Interestingly enough, the joint optimization of the step size and the batch size lead to a coordinate descent algorithm, where only one component (the one associated with the largest absolute-value policy gradient estimate) of the policy parameter gets updated. This algorithm is proven to converge as shown in [30]. Moreover, the resulting step size becomes a constant and the main role is played solely by the adaptive batch size.

3.6 Trust Region Policy Optimization

Following the results of Kakade & Langford, Shulman et al. [26] address the problem of safe algorithms going too conservative. In fact, it is really hard to develop a tight bound on policy performance improvement since there are several stochastic factors coming into play and the variance of the gradient estimates can be very high.
As shown in Section 3.5, a possibility is to adapt the step size and the batch size to guarantee safe updates while maintaining the cost per trajectory as low as possible. However this does not fully overcome the conservativeness of such approaches, because, as we have seen in Section 3.4, the practical bound from Theorem 3.10 originates from a series of simplifications and upper bounds on hard-to-estimate quantities, such as the infinite norm of the action-value function $\|Q^{\pi_{\boldsymbol{\theta}}}\|_\infty$.

Shulman et al. try to overcome this issue starting from the results of Kakade & Langford and proposing a new algorithm called Trust Region Policy Optimization (TRPO). This new algorithm is intended to give monotonic performance improvements and can be used on large tasks such as Atari games, at the cost of some hyper-parameter tuning. However, the practical version of the algorithm proposed by Shulman et al. suffers from several approximations that make it lose most safety properties.
The derivation of the bound starts from the following insight, proven in [23]:

Lemma 3.15. *For any pair of policies* π *and* $\tilde{\pi}$*, the performance of* $\tilde{\pi}$ *can be written*

Algorithm 3.2 Policy iteration algorithm guaranteeing non-decreasing expected return J

```
Initialize $\pi_0$
for i=0,1,2,... until convergence do
    Compute all advantage values $A_{\pi_i}(s,a)$
    Solve the constrained optimization problem:
    $\pi_{i+1} = \arg\max_\pi [L_{\pi_i}(\pi) - CD_{KL}^{\max}(\pi_i,\pi)]$
        where $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$
        and $L_{\pi_i}(\pi) = J(\pi_i) + \frac{1}{1-\gamma}\sum_s d_\mu^{\pi_i}(s)\sum_a \pi(a|s)A_{\pi_i}(s,a)$
end for
```

as a function of the advantage A_π*:*

$$J_\mu(\tilde{\pi}) = J_\mu(\pi) + \mathbb{E}_{s_0,a_0,\ldots,\sim\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t)\right] =$$
$$= J_\mu(\pi) + \frac{1}{1-\gamma}\sum_s d_\mu^{\tilde{\pi}}(s)\sum_a \tilde{\pi}(a|s)A_\pi(s,a).$$

A first approximation done by the authors is to remove the dependency from the future state distribution of the policy $\tilde{\pi}$ that is unknown and replace it with the future state distribution of the current policy π:

$$d_\mu^{\tilde{\pi}} \sim d_\mu^{\pi} \tag{3.15}$$

Next, they defined the quantity L_π, a local approximation of the policy performance $J(\pi)$:

$$L_\pi(\tilde{\pi}) = J(\pi) + \frac{1}{1-\gamma}\sum_s d_\mu^{\pi}(s)\sum_a \tilde{\pi}(a|s)A_\pi(s,a).$$

Note. When the policy π is a parameterized by θ and is differentiable, with the approximation from Equation (3.15), we still have that $L_{\pi_{\theta_0}}$ is a first-order approximation of the performance $J(\pi_{\theta_0})$ evaluated in θ_0

In this setting, the authors derived the following result:

Theorem 3.16. *Let $D_{KL}^{\max}(\pi,\tilde{\pi}) = \max_s D_{KL}(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s))$ be the maximum Kullback-Lieber divergence between two policies π and $\tilde{\pi}$. Then, the following bound holds:*

$$J(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - CD_{KL}^{\max}(\pi,\tilde{\pi}),$$

where $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$ and $\epsilon = \max_{s,a}|A_\pi(s,a)|$.

Note that this result is similar (and slightly weaker) to the result from Kakade shown in Theorem 3.1. There are again two terms, the first related to the advantage of policy $\tilde{\pi}$ with respect to π and the second term is a penalization given by the

Kullback-Lieber divergence between the two policies.
A policy iteration algorithm can be obtained by observing that:

$$J(\pi_{i+1}) - J(\pi) \geq M_i(\pi_{i+1}) - M_i(\pi_i),$$

where $M_i(\pi) = L_{\pi_i} - CD_{KL}^{\max}(\pi_i, \pi)$.

Algorithm 3.2 shows a proposed implementation of a policy iteration algorithm that uses the new bounds defined in Theorem 3.16. However, this algorithm is not practical to use in real scenarios, as it requires to know some quantities such as $D_{KL}^{\max}$ that is difficult to approximate. Moreover, this algorithm would yield very small step sizes, nullifying the main objective of this work. Hence, starting from this general algorithm, the authors proposed a more practical version that comes with several approximations, listed below:

- First of all, they considered parameterized policies $\pi_{\boldsymbol{\theta}}$.
- Instead of maintaining a penalty cost on the KL divergence in the objective function, they changed the problem formulation by maximizing the advantage and putting the KL divergence between the two policies as a constraint. In this way the problem becomes a constrained optimization problem on the policy parameters $\boldsymbol{\theta}$.
- They replaced the maximum of the KL divergence with the average KL divergence, which is easier to estimate.
- They provided approximation techniques to estimate the objective and the constraint from experience samples.

The above approximations lead to an algorithm that is very effective in practice, yielding very good results even in complex domains such as Atari games. However, it is also proven to be very sensitive to noise and require hyper-parameter tuning to be effective, suffering from high variance in performance measure overall [31].

3.7 Alternative Safety Approaches

All the methods considered so far try to achieve monotonic improvement. Following Definition 3.1, monotonic improvement is a special case of safety constraint where the reference performance is equal to the performance of the policy at the previous iteration. This constraint is not always mandatory in practice. In many applications, for example, we can allow for some non-monotonic updates as long as the policy does not perform worse than a given reference (e.g., the performance of the initial policy). In this case we can set a lower-bound on the performance $\underline{J_\mu}$ to be equal to the performance a user-defined policy. This looser constraint allows for more exploration, thus yielding more promising results.

This is the line of research followed by Thomas et al. [32], who provided an algorithm that, given some user-defined policy performance $\underline{J_\mu}$ and a factor δ, yields a policy π_{new} whose performance $J_\mu(\pi_{new})$ is no worse than $\underline{J_\mu}$ with probability $1 - \delta$.
The proposed batch RL algorithm splits the sample trajectories in a train and a test set. The new policy π_{new} is searched using the train set among the policies considered safe, while the test set is used to evaluate the quality of the new policy using offline policy evaluation with importance sampling [33].

Ghavamzadeh et al. [34] presented a different approach to safe policy improvement. They considered a model-based approach, where a model of the MDP is built upon samples collected from experience. In order to model the uncertainty of the environment dynamics, the authors considered a robust regret minimization problem, where the new proposed policy π_{new} is robustly evaluated (i.e. by adding noise) against the approximated model to minimise the negative regret w.r.t. a baseline policy.

In this chapter we have seen some approaches to safe learning. Monotonic improvements described in Sections 3.2-3.6 often suffer to be too conservative, that is why the looser constraints described in this section are taken into considerations. However these latter approaches are not applied to policy gradient methods, which offers nice theoretical convergence properties, at least to a local maximum. The focus of this work is to extend the results seen in Section 3.4 in policy gradient to non-monotonic safety constraints. This was done by first introducing a general framework for safe learning and then developing new theoretical results, as described in details in Chapter 4.

Chapter 4

Balancing Safety and Exploration in Policy Gradient

One of the challenges that arise in reinforcement learning - opposed to other kinds of learning - is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent can favour actions that it has tried in the past and found to be effective in producing reward. At the same time, though, discovering such actions requires a trial-and-error search. The agent has to exploit what it has already experienced in order to obtain reward, but it also has to explore in order to make better action selections in the future. This dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task: a full-exploit algorithm would likely choose sub-optimal actions while a fully exploratory behaviour will likely to take random actions without long-time awareness. The agent must try a variety of actions and progressively favour those that appear to be the best. This dilemma can be made even harder when we consider stochastic rewards, where the agents must try the same actions multiple times to be sure about their estimated values. The exploration-exploitation dilemma has been studied intensively for many decades, yet remains an open problem.

In this chapter, we will combine this important aspect of exploration with safe learning. As it is shown in Chapter 3, safe learning algorithms tend to be over-conservative, resulting in slow and inefficient learning. By customizing safety constraints to match the user needs, we can allow for a controlled exploration that is likely to speed up convergence. The final result of this chapter is a new practical safe learning algorithm named Safely-Exploring Policy Gradient (SEPG).

Section 4.1 will start by describing the problem of exploration in more details and analysing previous approaches for policy gradient. Then, we will describe a new general framework in Section 4.2 that reconciles all the aspects of safe learning; in this framework, we will also list special cases corresponding to relevant application scenarios. In Section 4.3 we will generalize existing performance improvement bounds for Gaussian parametric policies in two ways: we will take more general safety constraints into consideration, and we will deal with adaptive policy variance. Furthermore, we will propose a more exploratory policy update. Finally, in Section 4.4 we will exploit this theoretical analysis to build policy gradient algorithms tailored to the different safe-learning scenarios, providing Safely-Exploring Policy Gradient (SEPG), a very general algorithm that can be customized to one's needs. Comparative results of SEPG on benchmark continuous RL tasks will be analysed in detail in Chapter 5.

4.1 Exploration in Safe Policy Gradient

Exploration is a fundamental aspect of RL (agents that do not value exploration are often excessively greedy, tending to get stuck in sub-optimal behaviours) and is a topical object of study in the recent literature. At the same time, an excessive exploratory behaviour can harm systems and people (e.g., an excessive movement can lead to the breaking of a robotic arm), or have an intuitive economical impact (e.g., the cost of substituting the robotic arm).

In this work, we aim to find a good trade-off between these competing aspects, exploration and safety, using policy gradient method. Previous approaches in safe policy gradient, described in Section 3.4, employ a Gaussian policy with constant parameter σ that regulates the amount of exploration. However, the optimal value of σ is often domain-dependent and previous approaches do not take into account the possibility of adapting σ to the surrounding environment.
One possibility to extend this approach is to update σ like one of the policy parameters: $\sigma \leftarrow \sigma + \nabla_\sigma J_\mu(\pi_{\boldsymbol{\theta},\sigma})$. However this approach has some limitations:

- The bounds on policy improvement in [4] are no longer applicable. For this reason, we are not guaranteed to have safe updates.
- In many cases (e.g., LQG) exploration is highly penalized. Taking an action different than the optimal one, in fact, usually reduces the performance of the policy. We will refer to the reduction of performance due to exploratory behaviour as the *exploration cost*. By following the gradient w.r.t. σ of the performance $J_\mu(\boldsymbol{\theta})$, the agent usually tends to reduce the exploration cost, resulting in a policy that is no longer exploratory, but almost deterministic. This is not good since a lack of exploration usually causes a really slow learning process and it makes it hard to escape from local maxima.

A different approach was taken by [35], [36], [37], who employed maximum-entropy reinforcement learning to improve exploration strategy. A new intrinsic reward is given to the agent proportional to the entropy of the action distribution in that state. In this way, the agent is encouraged to explore as a way to increase the entropy of the policy. Despite its good result in practice, there are still some aspects that should be taken into consideration:

- The agent is optimising a surrogate objective function that is not defined by the problem.
- This method introduces an additional hyper-parameter (the weight of the entropy intrinsic reward) that has to be tuned for each environment.
- It is not clear how to guarantee safe updates with a surrogate objective function.

In the rest of this chapter, we will propose a new method to adapt exploration in safe reinforcement learning by setting σ so as to maximise a guaranteed lower-bound on future performance improvement.

4.2 Safe Learning Framework

As we have seen in Chapter 3, several works studied the problem of safety in reinforcement learning mostly referring to the monotonic improvement case. In this section, we will present a new framework that will serve as a starting point also for other safety constraints, like the ones defined in Section 3.7. Specifically, we identify two main categories of safety constraints (type-I and type-II) that differ from their meaning in practical applications.

4.2.1 Type-I safety

Type-I safety guarantees that the agent never performs dangerous behaviours. This can be imposed by explicitly defining dangerous states and/or actions [38] or by designing the reward signal such that danger is matched with low performance values [3] [26] [34].
This can be accomplished by imposing a condition of the kind:

$$J(\boldsymbol{\theta}^{t+1}) \geq J_B^t \tag{4.1}$$

for every time step t with respect to a certain baseline J_B^t. The baseline represents a lower-bound on the acceptable policy performance.
This condition can be achieved by constraining each parameter update as follows:

$$J(\boldsymbol{\theta}^{t+1}) - J(\boldsymbol{\theta}^t) \geq C^t, \tag{4.2}$$

for some $C^t \in \mathbb{R}$, where $J_B^t = C^t + J(\boldsymbol{\theta}^t)$. Depending on the sign of C^t, we give the following definitions:

Definition 4.1 (Required improvement)**.** We define an update in 4.2 to be a *required improvement* when $C^t \geq 0$. A required improvement happens when we are forced to improve the policy performance by a given quantity C^t.

Definition 4.2 (Bounded worsening)**.** We define an update in 4.2 to be a *bounded worsening* when $C^t < 0$. A bounded worsening happens when we can afford to lose at most $|C^t|$ in a single update. The typical case is when we want to exit from a local maximum but we put a constraint on the performance loss.

The value of the baseline policy J_B^t and, in particular, the value of C^t, can be used to ensure different properties. Two interesting settings that can be identified are:

Monotonic Improvement (MI): Setting $C^t = 0$ forces a monotonic improvement behaviour, which models the case in which the agent is constrained to continually improve its performance. This is appropriate when policy updates can be done rarely, take a lot of time or have costs/risks that are not modelled in the reward. This is the constraint considered in the safe policy gradient literature presented in Chapter 3.

Lower-Bounded I (LB-I): Setting $C^t = J(\boldsymbol{\theta}^0) - J(\boldsymbol{\theta}^t)$ puts a lower bound on the performance that matches the performance of the initial policy $\pi_{\boldsymbol{\theta}^0}$. In this case the main goal is to never do worse than the initial policy. This is appropriate when the initial policy have been designed to avoid dangerous behaviour (modelled in the reward), or when improvement over an already good policy must be guaranteed to justify the deployment of an RL algorithm [32].

4.2.2 Type-II safety

Type-II safety has an economic connotation. Here the reward is designed to record costs and revenues and the constraint is to have sufficient earnings (or limited losses) over some significant horizon. Low performance (e.g., due to exploration) is acceptable as long as it is repaid soon enough by an improved behaviour [5] [32].
Formally, type-II safety can be defined so as to guarantee a minimum *average* performance J_B^t over some interesting time horizon, that we identify with a *learning iteration*. This could be, e.g., a day of production in an automated factory, an accounting trimester and so on. We can run different policies within an iteration, as long as the average performance is above a certain baseline J_B^t.
We can formalize it as:

$$\frac{1}{K}\sum_{k=1}^{K} J(\boldsymbol{\theta}^{k(t)}) \geq J_B^t, \tag{4.3}$$

where $\boldsymbol{\theta}^{k(t)}$ denotes one of K policies that are evaluated within iteration t. This is weaker than type-I, since we have no guarantee on the performance of the single policies that are evaluated – some of them may perform arbitrarily bad, as long as others are able to compensate. The K policies may represent, e.g., intermediate updates to subsets of parameters (as in Section 4.3) or multiple updates within a learning iteration.

Even more interestingly, we can use constraint (4.3) to model the following scenario: assume we want to optimize a target policy $\pi_{\boldsymbol{\theta}_T}$ while running a (possibly different)

learning policy $\pi_{\boldsymbol{\theta}_E}$. For instance, we may want to find an optimal deterministic controller, while employing a stochastic policy to explore original solutions (as in Deterministic PG algorithms [39]).
We will always refer to the exploratory policy parameters with $\boldsymbol{\theta}_E$ and to the target policy parameters with $\boldsymbol{\theta}_T$. In this case, we can restate Equation (4.3) in this way:

$$(J(\boldsymbol{\theta}_E^t) + J(\boldsymbol{\theta}_T^t))/2 \geq J_B^t \tag{4.4}$$

Note. The bound in Equation (4.3) can be obtained by setting $\boldsymbol{\theta}_T \equiv \boldsymbol{\theta}_E$. In this case, we are optimizing the performance of the learning policy $\pi_{\boldsymbol{\theta}_E}$ and we are not evaluating any target policy.
Also, note that evaluating the target policy is not mandatory, but may provide useful information on the quality of the current solution.

Again, this is equivalent to constraining each parameter update to have the following form:

$$J(\boldsymbol{\theta}_E^{t+1}) - J(\boldsymbol{\theta}_E^t) + J(\boldsymbol{\theta}_T^{t+1}) - J(\boldsymbol{\theta}_T^t) \geq C^t, \tag{4.5}$$

for some $C^t \in \mathbb{R}$. In the spirit of type-II safety, we define a quantity B, called *budget*, defined as follows:

Definition 4.3 (Budget)**.** We denote the budget to be the cumulative sum of all the policy performance changes over a learning iteration

$$B^t = \sum_{k=0}^{t} J(\boldsymbol{\theta}_E^t) - J(\boldsymbol{\theta}_E^{t-1}) + J(\boldsymbol{\theta}_T^t) - J(\boldsymbol{\theta}_T^{t-1}). \tag{4.6}$$

This quantity has an intuitive economical meaning: it includes all the earnings that we got so far through our policy optimization process.
When $B^t > 0$, it means that our current policy performs, on average, better than the baseline, so we can potentially yield a worse policy without breaking the constraint (bounded worsening of at most B^t units).
When $B^t < 0$, instead, it means that the current policy is in average worse than the baseline, so we are required to take safer updates and improve the performance (required improvement of at least $|B^t|$ units).

Note. In general we want the budget to be non-negative. However approximation errors could bring to situations where the budget is negative. If this is the case, the intended behaviour would be to recover the budget as soon as possible via required improvements.

Type-II safety can be restated as a budget constraint as follows:

Theorem 4.1. *Any iterative learning algorithm that maintains $B^t \geq 0$ fulfills type-II safety with respect to a constant baseline $J_B = [J(\boldsymbol{\theta}_E^0) + J(\boldsymbol{\theta}_T^0)]/2$.*

We identify three special cases of type-II safety corresponding to different choices of target policy:

Table 4.1: Overview of the safety requirements outlined in Section 4.2.

Name	Target policy $\boldsymbol{\theta}_T^t$	Baseline J_B^t	C^t	Type
MI	$\boldsymbol{\theta}_E^t$	$J(\boldsymbol{\theta}_E^t)$	0	I
LB-I	$\boldsymbol{\theta}_E^t$	$J(\boldsymbol{\theta}_E^0)$	$J(\boldsymbol{\theta}_E^0) - J(\boldsymbol{\theta}_E^t)$	I
LB-II	$\boldsymbol{\theta}_E^t$	$J(\boldsymbol{\theta}_E^0)$	$-B^t$	II
LV	$\tilde{\sigma}^t < \sigma^t$	$[J(\boldsymbol{\theta}_E^0) + J(\boldsymbol{\theta}_T^0)]/2$	$-B^t$	II
ZV	$\tilde{\sigma}^t = 0$	$[J(\boldsymbol{\theta}_E^0) + J(\boldsymbol{\theta}_T^0)]/2$	$-B^t$	II

Lower-Bounded II (LB-II): When $\boldsymbol{\theta}_T \equiv \boldsymbol{\theta}_E$, we aim to maximise the online performance without doing worse than the initial policy, in terms of average iteration performance. This is a common online-learning scenario.

Low-Variance (LV): Here $\boldsymbol{\theta}_T$ is stochastic, but less exploratory than $\boldsymbol{\theta}_E$. An example scenario is the case in which learning is performed on a prototype before testing on the real system, which may be more critical.

Zero-Variance (ZV): Here $\boldsymbol{\theta}_T$ is deterministic. This controller gives more realistic results, but, as we will see in the next section, it is more challenging to give guarantees on deterministic policies.

The safety requirements that we have identified are summarised in Table 4.1. In Section 4.4 we will propose a general algorithm that can be customised to match any of these constraints.

4.3 Theoretical Analysis

In this section we will develop new theoretical results that will be the main building blocks of the SEPG algorithm outlined in Section 4.4. In particular, we will first extend the results obtained by Pirotta et al. [4], explained in Section 3.4, to deal with an adaptive exploration parameter σ. Next, we will derive a new way to update σ to allow for more exploration while satisfying safety constraints.

4.3.1 Preliminaries

In this theoretical analysis we will focus on the guaranteed improvement case where $\boldsymbol{\theta}_T^t = \boldsymbol{\theta}_E^t = \boldsymbol{\theta}^t$, so that the safety constraint can be expressed as:

$$J(\boldsymbol{\theta}^{t+1}) - J(\boldsymbol{\theta}^t) \geq C^t, \quad C^t \in \mathbb{R}, C^t < C_{\max}.$$

Here $C_{\max}$ represents the maximum guaranteed improvement allowed by the theoretical bound on performance improvement. Since all the bounds that we are going to use are second-order polynomials, the maximum can be easily found in closed-form. From this specific case, we will see how to generalise to all the other cases presented in Section 4.2.

Moreover, we will focus on Gaussian policies of the form:

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

using the following, common parametrisation:

$$\mu(s) = \boldsymbol{v}^T\boldsymbol{\phi}(s), \qquad \sigma = e^w, \qquad \boldsymbol{\theta} = [\boldsymbol{v}|w], \tag{4.7}$$

where $\boldsymbol{\phi}$ are bounded state-features, i.e., $\phi_i(s) \leq M_{\boldsymbol{\phi}}$ for each $s \in \mathcal{S}, i = 1, \dots, m$. This parametrization is such that we have always a positive exploration coefficient $\sigma = e^w$ and the deterministic policy is approached when $w \to -\infty$. For this reason, we will define the deterministic policy with $\pi_{\boldsymbol{\theta}}$ where $\boldsymbol{\theta} = [\boldsymbol{v} \mid -\infty]^{\mathsf{T}}$.

More complex representations are possible, e.g., μ can be parametrized by a neural network. In some cases, also the standard deviation can be state-dependent. However, in this work we will only consider the linear-mean parametrization.
Another common generalization is to extend the policy to multi-dimensional action spaces by employing an independent Gaussian policy for each action dimension. This extension will be discussed in Section 4.3.4.

Given this new notation, we can restate and summarise the theorems described in Section 3.4 and Section 3.5, that will be the starting point for the adaptive variance case described in the next section:

Theorem 4.2 (From Theorem 3.3 in [5]). *Assuming $w^t = const$ for all t, update rule (4.11) guarantees:*

$$J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t) \geq \alpha \left\|\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t)\right\|_\infty^2 - \alpha^2 c \left\|\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t)\right\|_\infty^2, \tag{4.8}$$

where $c = \frac{RM_\phi^2}{(1-\gamma)^2\sigma^2}\left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)}\right)$ and $|\mathcal{A}|$ is the volume of the action space. Guaranteed improvement is maximised by using a step size $\alpha^* = \frac{1}{2c}$, yielding:

$$J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t) \geq \frac{\left\|\nabla_{\boldsymbol{\theta}} J(\boldsymbol{v}^t, w^t)\right\|_\infty^2}{4c} := C_{max}^{\boldsymbol{v}}. \tag{4.9}$$

Moreover, for any $C \leq C_{max}^{\boldsymbol{v}}$, the following constraints on the step-size α:

$$|\alpha - \alpha^*| \leq \lambda_{\boldsymbol{v}}\alpha^*, \tag{4.10}$$

where $\lambda_{\boldsymbol{v}} = \sqrt{1 - C/C_{max}^{\boldsymbol{v}}}$, are enough to guarantee $J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t) \geq C$.

For simplicity, we assume to be able to compute exact policy gradients and we focus on one-dimensional action spaces. We will deal with approximation errors in Section 4.5.

4.3.2 Safe PG with adaptive variance

As we have seen in Chapter 3, a way to guarantee performance improvement in policy gradient is to adaptively tune the step size α to avoid large parameter updates (overshooting). For the case of Gaussian policies with linear mean and fixed variance, Papini et al. [5] proposed a greedy-coordinate-ascent update:

$$v_k^{t+1} \leftarrow \alpha \nabla_{v_k} J(\boldsymbol{v}^t, w^t) \quad \text{if } k = \arg\max_i |\nabla_{v_i} J(\boldsymbol{v}^t, w^t)| \text{ else } v_k^t, \tag{4.11}$$

proving that it yields better guarantees than any gradient ascent update.
Only the parameter with maximum absolute-value partial derivative is updated, and ties can be broken in any way[1]. Greedy coordinate ascent does not inherit the convergence properties of coordinate ascent, but has been proven to converge anyway [30]. Theorem 4.2 shows the performance improvement properties of this algorithm.

We generalise this result to the adaptive-variance case. Since the above guarantees require a fixed w, we can split the update rule in two parts: we first update the mean parameter ($\boldsymbol{v}$-update) keeping w fixed, then we update the variance parameter (w-update) keeping the mean parameters $\boldsymbol{v}$ fixed:

$$\begin{cases} v_k^{t+1} \leftarrow v_k^t + \alpha \nabla_{v_k} J(\boldsymbol{v}^t, w^t) & \text{if } k = \arg\max_i |\nabla_{v_i} J(\boldsymbol{v}^t, w^t)| \text{ else } v_k^t, \\ w^{t+1} \leftarrow w^t + \beta \nabla_w J(\boldsymbol{v}^{t+1}, w^t), \end{cases} \tag{4.12}$$

where β is a step size for the variance parameter. Since, in general, the intermediate policy $\pi(\cdot|\boldsymbol{v}^{t+1}, w^t)$ must be evaluated in order to compute $\nabla_w J(\boldsymbol{v}^{t+1}, w^t)$, we actually need to divide the required improvement C between the two updates:

$$J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t) \geq (1-\nu)C \quad \text{and} \tag{4.13}$$

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \nu C, \tag{4.14}$$

with any coefficient $\nu \in [0, 1]$.

The first constraint (4.13) is already covered by Theorem 4.2, by replacing C with $(1-\nu)C$, since σ stays constant during the $\boldsymbol{v}$-update.
Addressing the problem of establishing similar performance improvement properties for the w-update in (4.14), we derive the following result:

Theorem 4.3. *Assuming $o(|\Delta w|^3)$ terms can be neglected,*[2] *the w-update in (4.12) guarantees:*

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \beta \nabla_w J(\boldsymbol{v}^{t+1}, w^t)^2 - \beta^2 d \nabla_w J(\boldsymbol{v}^{t+1}, w^t)^2, \tag{4.15}$$

[1] It is important to choose a single parameter in case of ties, otherwise convergence is not guaranteed

[2] This approximation is not critical since the steps produced by safe algorithms tend to be very small.

where $d = \frac{R}{(1-\gamma)^2}\left(\frac{\psi|\mathcal{A}|}{2\sigma_w} + \frac{\gamma}{1-\gamma}\right)$ *and* $\psi = 4(\sqrt{7}-2)\exp(\sqrt{7}/2-2)/\sqrt{2\pi} \simeq 0.524$. *Guaranteed improvement is maximised by using a step size* $\beta^* = \frac{1}{2d}$*, yielding:*

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \frac{\nabla_w J(\boldsymbol{v}^{t+1}, w^t)^2}{4d} := C^w_{max}. \tag{4.16}$$

Moreover, for any $\nu C \leq C^w_{max}$*, the following constraints on the step-size* β*:*

$$|\beta - \beta^*| \leq \lambda_w \beta^*, \tag{4.17}$$

where $\lambda_w = \sqrt{1 - \nu C / C^w_{max}}$*, are enough to guarantee* $J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \nu C$*.*

Given this result, we can safely adapt the variance of a Gaussian policy w while keeping $\boldsymbol{v}$ fixed. By combining the results in Theorem 4.2 and Theorem 4.3 we can safely update the mean and variance of a Gaussian policy $\pi_{\boldsymbol{\theta}}$ by applying a $\boldsymbol{v}$-update (4.13) and a w-update (4.14) in cascade.
However, as stated in Section 4.1, a greedy update of w may not be a good choice because of its tendency to reduce exploration costs, that would quickly turn w to low values. The next section will provide a solution to this problem.

4.3.3 Safely Exploring PG

The w-update in (4.12) does not take into consideration, in any way, the potential benefits of increasing exploration. In practice, following $\nabla_w J$ often results in greedy learning, setting the policy variance to small values very soon in order to exploit the current solution, which is usually suboptimal. More far-sighted algorithms would keep the variance high, or even increase it, long enough to properly explore the space of policies.
We now provide a variant of (4.12) that is more exploratory while preserving safety guarantees. The first key intuition is that, in many situations, larger values of $\sigma = e^w$ can positively affect the guaranteed improvement yielded by the next $\boldsymbol{v}$-update. Hence, we want to update w in order to maximise this improvement.
We first need a simplified version of Theorem 4.2:

Corollary 4.4. *The* $\boldsymbol{v}$*-update in (4.12) with optimal step size* α^* *from Theorem 4.2 guarantees:*

$$J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t) \geq \frac{\left\|\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t)\right\|_2^2}{4mc} := \mathcal{L}(\boldsymbol{v}^t, w^t),$$

where $\mathcal{L}$ is a looser, but differentiable, lower bound on performance improvement for the $\boldsymbol{v}$-update. Note that the step size α used to update $\boldsymbol{v}$ is not relevant[3], so we used the optimal step size α^* provided in [5] instead of the one used for the actual

[3]We are making an optimistic assumption here, but any value of α can be used without breaking the safety constraint

$\boldsymbol{v}$-update.

We then use $\mathcal{L}$ as a surrogate objective for the w-update:

$$\begin{cases} v_k^{t+1} \leftarrow v_k^t + \alpha \nabla_{v_k} J(\boldsymbol{v}^t, w^t) & \text{if } k = \arg\max_i |\nabla_{v_i} J(\boldsymbol{v}^t, w^t)| \text{ else } v_k^t, \\ w^{t+1} \leftarrow w^t + \beta \nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t). \end{cases} \tag{4.18}$$

The idea is to update w so as to maximise the performance improvement that the subsequent $\boldsymbol{v}$-update is able to guarantee. This is a more far-sighted version of (4.18), using the bound in Corollary 4.4 to look one step ahead. Since high values of w lead to better performance improvement bounds, this typically favours exploration over exploitation.

It remains to establish the safety of this new update. The next key insight is that Theorem 4.3 provides a measure of guaranteed improvement for any step size β, even negative ones. So, even if the w-update is in the direction of $\nabla_w \mathcal{L}$, we can measure guaranteed improvement by the corresponding step in the direction of $\nabla_w J$ with a simple rescaling:

Theorem 4.5. *The w-update in (4.18) guarantees, for any (possibly negative) step size β:*

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \beta \nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t) \nabla_w J(\boldsymbol{v}^{t+1}, w^t) - \beta^2 d \nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t)^2.$$

Guaranteed improvement is maximised by using a step size $\tilde{\beta} = \frac{\nabla_w J(\boldsymbol{v}^{t+1}, w^t)}{2d \nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t)}$, yielding:

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq C^w_{max}.$$

Moreover, for any $\nu C \leq C^w_{max}$, the following constraints on the step-size β:

$$|\beta - \tilde{\beta}| \leq \lambda_w |\tilde{\beta}| \tag{4.19}$$

are enough to guarantee $J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \nu C$.

Finally, to further encourage parameter-space exploration, we propose to use the largest step sizes that satisfy the safety guarantees. This is natural for the w-update, that is ancillary to the following $\boldsymbol{v}$-update. As for the $\boldsymbol{v}$-update, a larger step size can substantially improve convergence speed. In fact, an improvement in the $\boldsymbol{v}$-update can obtain some budget to be spent on exploration in the subsequent w-update. A more efficient exploration can yield a more accurate estimation for the next $\boldsymbol{v}$-update and so on, resulting in a positive feedback loop.

From Theorem 4.2 and Theorem 4.5, respectively, the largest safe $\boldsymbol{v}$-step size is $\overline{\alpha} := (1 + \lambda_{\boldsymbol{v}})\alpha^*$, and the largest safe w-step size is $\overline{\beta} := \tilde{\beta} + \lambda_w |\tilde{\beta}|$. Hence, our proposed update rule is:

$$\begin{cases} v_k^{t+1} \leftarrow v_k^t + \overline{\alpha} \nabla_{v_k} J(\boldsymbol{v}^t, w^t) & \text{if } k = \arg\max_i |\nabla_{v_i} J(\boldsymbol{v}^t, w^t)| \text{ else } v_k^t, \\ w^{t+1} \leftarrow w^t + \overline{\beta} \nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t), \end{cases} \tag{4.20}$$

which combines greediness and exploration while satisfying Safety Constraint (4.13, 4.14) for any $C \leq \min\{C^{\boldsymbol{v}}_{\max}, C^w_{\max}\}$ and $\nu \in [0, 1]$.

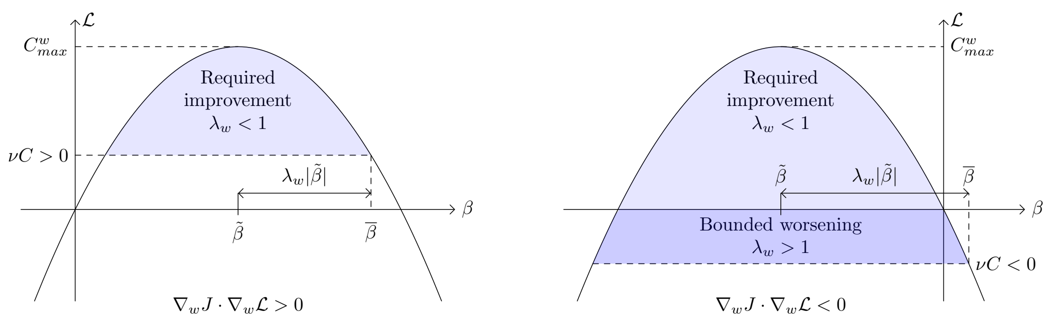


Figure 4.1: The step size $\overline{\beta}$ in (4.20) can be in the direction of $\nabla_w J(\boldsymbol{v}, w)$, in which case it will guarantee an improvement, or can be in the opposite direction (in the direction of $\nabla_w \mathcal{L}(\boldsymbol{v}, w)$) when C is negative, guaranteeing a bounded worsening.

Figure 4.1 recaps the idea of this novel update. By considering an update rule for the w-update of this form: $w^{t+1} \leftarrow w^t + \overline{\beta}\nabla_w J_\mu(\boldsymbol{v}^{t+1}, w^t)$ we can have two cases:

- $\nabla_w J(\cdot) \cdot \nabla_w \mathcal{L}(\cdot) > 0$: The update direction of the surrogate objective is the same as the naïve one. In this case the step size $\overline{\beta}$ will be positive and will follow both the surrogate and the naïve gradient.
- $\nabla_w J(\cdot) \cdot \nabla_w \mathcal{L}(\cdot) < 0$: In this case we are likely to move in the opposite way w.r.t. the naïve gradient $\nabla_w J_\mu(\cdot)$. More specifically, we can identify two sub-cases:
  - If we must guarantee a required improvement, then the step size $\overline{\beta}$ will be negative and we move against the surrogate objective and towards the naïve gradient direction.
  - Otherwise, if we can afford a bounded worsening, the step size $\overline{\beta}$ will be positive and we move according to the surrogate objective and against the naïve gradient. This is the most interesting case where the w-update is in the opposite direction w.r.t. conventional updates.

4.3.4 Multi-dimensional actions

In this section, we extend the results above to the more general case of multi-dimensional action spaces. As mentioned in Section 2.1, a common policy class for the case $\mathcal{A} \in \mathbb{R}^l$ is the factored Gaussian, i.e., a multi-variate Gaussian distribution having a diagonal covariance matrix $\Sigma_{\boldsymbol{\theta}}$. We denote with $\boldsymbol{\sigma}_{\boldsymbol{\theta}}$ the vector of the diagonal elements, i.e., $\Sigma_{\boldsymbol{\theta}} = diag(\boldsymbol{\sigma}_{\boldsymbol{\theta}})$. So, with a little abuse of notation[4], we can write the factored Gaussian policy as:

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

[4]This allows us to avoid the much more cumbersome matrix notation, where even $\boldsymbol{v}$ is a matrix.

where all vector operations are component-wise. The result is, of course, a multi-dimensional action. The natural generalisation of Parametrisation (4.7) is:

$$\mu_{\boldsymbol{\theta}}(s) = \boldsymbol{v}^T \boldsymbol{\phi}(s), \qquad \boldsymbol{\sigma}_{\boldsymbol{\theta}} = e^{\boldsymbol{w}}, \qquad \boldsymbol{\theta} = [\boldsymbol{v}|\boldsymbol{w}], \tag{4.21}$$

where $\boldsymbol{w}$ is an l-dimensional vector. Following what [5] do for the mean parameter, we update $\boldsymbol{w}$ by greedy coordinate descent as well. All the results on $\boldsymbol{v}$ naturally extend to $\boldsymbol{w}$ since the bounds in Theorems 4.3 and 4.5 differ from the one in 4.2 only by a constant. We just provide the multi-dimensional version of Update 4.20:

$$\begin{cases} v_k^{t+1} \leftarrow \overline{\alpha} \nabla_{v_k} J(\boldsymbol{v}^t, w^t) & \text{if } k = \arg\max_i |\nabla_{v_i} J(\boldsymbol{v}^t, w^t)| \text{ else } v_k^t, \\ w_h^{t+1} \leftarrow \overline{\beta} \nabla_{w_h} \mathcal{L}(\boldsymbol{v}^{t+1}, w^t) & \text{if } h = \arg\max_j |\nabla_{w_j} \mathcal{L}(\boldsymbol{v}^t, w^t)| \text{ else } w_h^t. \end{cases}$$

An even further generalization would be to consider a non-diagonal covariance matrix. This is interesting, but out of the scope of this work: here we study the effects of the variance on exploration, while a full covariance matrix also models correlations among action dimensions that may be useful to learn in some tasks. Another promising generalisation, left to future work, is represented by a state-dependent policy variance, which would allow a more powerful kind of exploration.

4.4 Algorithms

In this section we will describe in details SEPG (Safely Exploring Policy Gradient), a general algorithm that can be customized to match all the safety requirements presented in Table 4.1. To facilitate the description of this algorithm, we will first consider the implementation of type-I safety constraints, where the target policy is the same as the exploratory policy (SEPG-I). Next, we will describe how to guarantee type-II constraint with SEPG-II and finally we will provide the complete algorithm that also considers a target policy which can be different than the exploratory one (SEPG).

4.4.1 SEPG-I

To implement type-I safety we employ the update rule in Equation (4.18) and we straightforwardly apply the results seen in Section 4.3 to compute the optimal step sizes $\overline{\alpha}$ and $\overline{\beta}$. Algorithm 4.1 shows a pseudocode for this case: we can divide the iteration in two blocks. The first block performs a $\boldsymbol{v}$-update using the results provided by Theorem 4.2. The second block performs a w-update using the results provided in Theorem 4.5.
In particular, we can identify the following two scenarios:

Monotonic improvement (SEPG-MI): can be implemented by setting $C^t = 0$ in line 3 and any value of $\nu \in [0, 1]$. It is straightforward to see that the algorithm does indeed monotonically improve the solution.

Algorithm 4.1 Safely-Exploring Policy Gradient for type-I safety (SEPG-I)

```
1:  input: $\boldsymbol{\theta}^0 = [\boldsymbol{v}^0, w^0]$, $N$
2:  for t = 0,1,2 ... do
3:      initialize the value of $C^t$
4:      evaluate $J(\boldsymbol{v}^t, w^t)$, $\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t)$
5:      Compute largest $\overline{\alpha}$ that guarantees:
            $J(\boldsymbol{v}^t + \overline{\alpha}\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t), w^t) - J(\boldsymbol{v}^t, w^t) \geq (1-\nu)C^t$
6:      $\boldsymbol{v}^{t+1} \leftarrow \boldsymbol{v}^t + \overline{\alpha}\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t)$        ▷ $\boldsymbol{v}$-update
7:
8:      evaluate $J(\boldsymbol{v}^{t+1}, w^t)$, $\nabla_w J(\boldsymbol{v}^{t+1}, w^t)$, $\nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t)$
9:      Compute largest $\overline{\beta}$ that guarantees:
10:
            $J(\boldsymbol{v}^{t+1}, w^t + \overline{\beta}\nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t)) - J(\boldsymbol{v}^{t+1}, w^t) \geq \nu C^t$
11:     $w^{t+1} \leftarrow w^t + \overline{\beta}\nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t)$        ▷ $w$-update
12: end for
```

Lower-Bounded I (SEPG-LB-I): can be implemented by setting $C^t = J(\boldsymbol{v}^0, w^0) - J(\boldsymbol{v}^t, w^t)$ and any value of $\nu \in [0,1]$ such that: $(1-\nu)C^t \leq C^{\boldsymbol{v}}_{\max}$ and $\nu C^t \leq C^w_{\max}$. It can be easily shown that this is enough to guarantee that the performance of every evaluated policy $J(\pi_{\boldsymbol{\theta}^t})$ will not be worse than the reference $J_B = J(\pi_{\boldsymbol{\theta}^0})$. In fact, the successive application of the $\boldsymbol{v}$-update and the w-update guarantees that:

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^t, w^t) \geq C^t,$$

that is, substituting the value of C^t:

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^t, w^t) \geq J(\boldsymbol{v}^0, w^0) - J(\boldsymbol{v}^t, w^t),$$

hence:

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) \geq J(\boldsymbol{v}^0, w^0).$$

Note. The $\boldsymbol{v}$-update and the w-update blocks (coloured in Algorithm 4.1) can be put in any order without breaking the safety guarantee. For the cases SEPG-MI and SEPG-LB-I listed above, the behaviour of the algorithm does not change.

4.4.2 SEPG-II

Type-I safety was straightforward to implement because it required a guarantee for each update, which was already proven by the theorems in Section 4.3. However, this is no longer true for type-II safety, which gives guarantees over a longer horizon, that we defined as learning iteration. For this reason we employ the budget B as defined in Equation (4.6). This is needed by the algorithm to have some memory of the past. The initial-budget parameter B^0 is useful for tasks that require more exploration and represents the initial cost that we are willing to pay to be able

to solve the task, but can be set to 0 in most cases. Then, the budget is updated with the measured performance improvement each time a policy is evaluated. These updates are carefully designed to guarantee $B > 0$ for every parameter update, so that we can employ Theorem 4.1 to guarantee type-II safety.
The pseudocode is shown in Algorithm 4.2. In particular we identify the following scenario:

Lower Bounded II (SEPG-LB-II): This case is entirely covered by Algorithm 4.2. In practice, the algorithm always guarantees $B \geq 0$ in every update, since in the $\boldsymbol{v}$-budget update, we ensure that the quantity $J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t \geq -B$. Adding this quantity to the budget would at most put B to 0. A similar intuition can be applied to the w-budget update.
Since the budget is always $B \geq 0$ for every update, the correctness of the algorithm is proven by Theorem 4.1.

Note. Algorithm 4.2 is an extension of Algorithm 4.1 as the latter can be seen as a special case where the budget is fixed ($B^t = -C^t$).

Note. Similarly to what we have outlined for Algorithm 4.1, the order of the $\boldsymbol{v}$ and w-update blocks can be freely reordered without breaking the algorithm guarantees. However in this case, since we employ a cumulative budget, the order of operations can change the behaviour of the algorithm. In particular, we have seen that performing the $\boldsymbol{v}$-update before the w-update leads to better results because the latter is likely to gain more from the earnings of the $\boldsymbol{v}$-update. In other words, the $\boldsymbol{v}$-update is likely to improve the performance of the policy, that is reflected in a positive budget increment. The w-update, instead, is likely to erode the budget to face the high exploration costs. By performing the updates in this order, the w-update can generally employ more budget, that will be spent in exploration. Depending on the task, this can result in a faster convergence of the parameters.

4.4.3 SEPG

Algorithm 4.3 shows the general pseudocode of Safely-Exploring Policy Gradient that, in addition to Algorithm 4.2, also considers type-II safety with a target policy. In this last version, we employ all the techniques used in Algorithm 4.2 and we introduce an additional block (showed in green) that will be evaluated when the target policy is different than the exploratory policy $\boldsymbol{\theta}_T^t \neq \boldsymbol{\theta}_E^t$. This middle block runs the new target policy and updates the budget with its change in performance.

Note. Like in Algorithm 4.2, the three blocks in Algorithm 4.3 (in different colour) can be put in any order without breaking the safety guarantees. We decided to put these blocks in this order following the same considerations done for Algorithm 4.2: in this way, in fact, the last block that performs the w-update receives more budget, which is spent to explore the environment.

The evaluation of the target policy requires additional considerations. As stated in line 11, we should be able to evaluate the target policy without losing more than

Algorithm 4.2 Safely-Exploring Policy Gradient for type-II safety (SEPG-II)

```
1: input: $\boldsymbol{\theta}^0 = [\boldsymbol{v}^0, w^0]$, $N$, $B^0$
2: initialize: $B \leftarrow B^0$
3: evaluate $J_B^0 = J(\boldsymbol{v}^0, w^0)$
4: for t = 1,2 ... do
5:
6:     evaluate $J(\boldsymbol{v}^t, w^t)$, $\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t)$
7:     Compute largest $\overline{\alpha}$ that guarantees
           $J(\boldsymbol{v}^t + \overline{\alpha}\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t), w^t) - J(\boldsymbol{v}^t, w^t) \geq -B$
8:     $\boldsymbol{v}^{t+1} \leftarrow \boldsymbol{v}^t + \overline{\alpha}\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t)$        ▷ $\boldsymbol{v}$-update
9:     $B \leftarrow B + J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t)$.        ▷ $\boldsymbol{v}$-budget update
10:
11:    evaluate $J(\boldsymbol{v}^{t+1}, w^t)$, $\nabla_w J(\boldsymbol{v}^{t+1}, w^t)$, $\nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t)$
12:    Compute largest $\overline{\beta}$ that guarantees
           $J(\boldsymbol{v}^{t+1}, w^t + \overline{\beta}\nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t)) - J(\boldsymbol{v}^{t+1}, w^t) \geq -B$
13:    $w^{t+1} \leftarrow w^t + \overline{\beta}\nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t)$        ▷ $w$-update
14:    $B \leftarrow B + J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t)$.        ▷ $w$-budget update
15: end for
```

B budget units. The problem is that we have no guarantee on the target policy in general, which can perform arbitrarily bad. We faced this problem in two ways: the first solution will provide a stochastic target policy with some theoretical guarantees. The second solution, instead, will provide a more practical algorithm that has no theoretical guarantees, but was found to be safe enough and remarkably faster than the other solutions. The two solutions are presented below:

Low Variance (SEPG-LV): In this setting, we derive a target policy that aims to have lower variance than the exploratory policy on which we can still have guarantees on its performance. This can be the case in which we want to evaluate our policy on a prototype before starting the production. The low-variance policy $\pi_{\boldsymbol{\theta}^t_{\text{LV}}}$ is defined as follows:

$$\pi_{\boldsymbol{\theta}^t_{\text{LV}}} : \boldsymbol{\theta}^t_{\text{LV}} = [\boldsymbol{v}^t \mid w^t_{\text{LV}}]^{\mathsf{T}},$$
$$w^t_{\text{LV}} = \min \underline{w} \text{ such that } J(\boldsymbol{v}^t, \underline{w}) - J(\boldsymbol{v}^t, w^t) \geq -B.$$

Intuitively, the low-variance policy is the policy that shares the same mean $\boldsymbol{v}$ of the exploratory policy and has the lowest possible value of $\underline{w}$ that allows a bounded worsening of at most B. The value of $\underline{w}$ can be computed using constraints (4.17) from Theorem 4.3

Zero Variance (SEPG-ZV): In this second scenario, we consider as a target policy the deterministic policy with the same mean $\boldsymbol{v}$ of the exploratory policy: $\boldsymbol{\theta}^t_{\text{ZV}} = [\boldsymbol{v}^t \mid -\infty]^{\mathsf{T}}$. In this case we have no guarantee on the performance of the deterministic policy, but we note that, in case the budget goes negative, the algorithm

Algorithm 4.3 Safely-Exploring Policy Gradient (SEPG)

```
1:  input: $\boldsymbol{\theta}_E^0 = [\boldsymbol{v}^0, w^0]$, $T$, $N$, $B^0$
2:  initialize: $B \leftarrow B^0$
3:  evaluate $J_B^0 = \left(J(\boldsymbol{v}^0, w^0) + J(\boldsymbol{\theta}_T^0)\right)/2$
4:  for t = 1,2 ... do
5:
6:      evaluate $J(\boldsymbol{v}^t, w^t)$, $\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t)$
7:      Compute largest $\overline{\alpha}$ that guarantees
            $J(\boldsymbol{v}^t + \overline{\alpha}\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t), w^t) - J(\boldsymbol{v}^t, w^t) \geq -B$
8:      $\boldsymbol{v}^{t+1} \leftarrow \boldsymbol{v}^t + \overline{\alpha}\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t)$        ▷ $\boldsymbol{v}$-update
9:      $B \leftarrow B + J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t)$.        ▷ $\boldsymbol{v}$-budget update
10:     if $\boldsymbol{\theta}_T^t \neq \boldsymbol{\theta}_E^t$ then
11:         evaluate target policy $J(\boldsymbol{\theta}_T^t)$ without losing more than $B$ budget units
12:         $B \leftarrow B + J(\boldsymbol{\theta}_T^t) - J(\boldsymbol{\theta}_T^{t-1})$        ▷ target budget update
13:     end if
14:
15:         evaluate $J(\boldsymbol{v}^{t+1}, w^t)$, $\nabla_w J(\boldsymbol{v}^{t+1}, w^t)$, $\nabla_w \boldsymbol{\mathcal{L}}(\boldsymbol{v}^{t+1}, w^t)$
16:         Compute largest $\overline{\beta}$ that guarantees
            $J(\boldsymbol{v}^{t+1}, w^t + \overline{\beta}\nabla_w \boldsymbol{\mathcal{L}}(\boldsymbol{v}^{t+1}, w^t)) - J(\boldsymbol{v}^{t+1}, w^t) \geq -B$
17:         $w^{t+1} \leftarrow w^t + \overline{\beta}\nabla_w \boldsymbol{\mathcal{L}}(\boldsymbol{v}^{t+1}, w^t)$        ▷ $w$-update
18:         $B \leftarrow B + J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t)$.        ▷ $w$-budget update
19:     end for
```

will automatically try to compensate by performing required-improvement steps both for the $\boldsymbol{v}$ and w-update. For this reason SEPG-ZV can formally guarantee type-II safety only on an infinite number of iterations.

4.4.4 Further considerations

Algorithm 4.3 shows a complete pseudocode of SEPG. In this section we will make some considerations that are useful for a correct implementation of the algorithm.

First of all, we assumed that each iteration of the algorithm should collect N sample trajectories in total. In this work we have split these N trajectories evenly among the blocks. Several solutions could be developed by splitting the trajectories in different ways, e.g., by giving more weight to the $\boldsymbol{v}$-update rather than the w-update.

A second observation is that in the pseudocode the $\boldsymbol{v}$-budget update requires the value of $J(\boldsymbol{v}^{t+1}, w^t)$, that is only estimated at the beginning of the third block. A similar consideration can be done for the w-budget update in line 18. This problem can be solved simply by delaying the $\boldsymbol{v}$-budget update between lines 15 and 16, right after the estimation of $J(\boldsymbol{v}^{t+1}, w^t)$. This will not break the safety guarantee since the value of the budget is not used until line 16 to calculate $\overline{\beta}$. We decided to present the pseudocode in this form because it is easier and can be clearly subdivided into the three blocks.

Finally, we tried to develop also a new variant that, like SEPG-ZV, does not have any theoretical guarantees, but converges significantly faster than any other variant. This new version, named SEPG-SIM was built by merging the $\boldsymbol{v}$-update and the w-update blocks into a single block. In other words, we computed $\overline{\alpha}$ and $\overline{\beta}$ simultaneously using a single batch of trajectories. This was far more sample efficient, since it required only 2/3 of the samples per iteration, but it raised a new problem: how to spilt the budget between the $\boldsymbol{v}$-update and the w-update. Since this new problem was quite challenging, we decided to leave this for future works.

4.5 Approximate Framework

As we have seen in Chapter 2, when learning in a model-free scenario, we need to estimate the policy gradient from samples. The error on these estimations can harm the safety guarantees described in Section 4.2. In this section, we will provide a solution to mitigate this problem by deriving high-probability variants of the performance improvement bounds of 4.3, as done in [4] and [5].
In the following derivation, we will assume that an unbiased estimator is employed for the policy gradient $\nabla_{\boldsymbol{\theta}} J$, e.g., REINFORCE (2.1). The estimation of $\nabla_w \mathcal{L}$ is more involved, but we provide an unbiased estimator in Appendix B.
First, we need to bound the gradient estimation error in high probability, and this must be done for every gradient that we estimate (e.g., $\nabla_{\boldsymbol{v}} J, \nabla_w J, \nabla_w \mathcal{L}$). To keep things simple, we make the following, very general assumption:

Assumption 4.6. *For each gradient of interest $\nabla_{\boldsymbol{\theta}} F$, there is an estimator $\widehat{\nabla}^N_{\boldsymbol{\theta}} F$ using batches of N samples and an $\epsilon_i > 0$ such that:*

$$\left|\nabla_{\theta_i} F - \widehat{\nabla}_{\theta_i} F\right| \leq \epsilon \quad \textit{with probability at least } (1-\delta) \textit{ for every } i = 1, \dots, m \textit{ and} \tag{4.22}$$

$$\left\|\widehat{\nabla}_{\boldsymbol{\theta}} F\right\|_\infty > \epsilon \quad \textit{always.} \tag{4.23}$$

The latter statement (4.23) can be used as a stopping condition in iterative algorithms. In fact, when no (estimated) gradient component is greater than noise ϵ, no reliable coordinate-ascent policy update can be performed. It remains to characterize the gradient estimation error to satisfy (4.23) in a useful and meaningful way. This can be done by employing known, distribution-independent statistical inequalities such as Chebyshev's, Hoeffding's or variants of Bernstein's inequality, as done in [5]. In this work, we choose to employ t-based confidence intervals to obtain less conservative high-probability bounds:

$$\begin{aligned}
\epsilon_i &:= t_{\delta/2,N-1}\sqrt{\frac{\widehat{\mathbb{V}\mathrm{ar}}\left[\widehat{\nabla}_{\theta_i}F\right]}{N}},\\
\epsilon &= \max_i\{\epsilon_i\},
\end{aligned}$$

where $\widehat{\mathbb{V}\mathrm{ar}}$ is the sample variance and $t_{\delta/2,N-1}$ is the $(1-\delta/2)$-quantile of a Student's t-distribution with $N-1$ degrees of freedom. Note that all the estimators that we employ are unbiased (REINFORCE for the policy gradient, the estimator proposed in Appendix B for $\nabla_w\mathcal{L}$). However, the underlying trajectory-generating process being clearly non-Gaussian, we need to invoke the Central Limit Theorem, i.e., our t-based confidence interval satisfies Assumption 4.6 only in the limit of an infinite batch size. We do not think this is critical since, in our experiments, we use $N \sim 100$, and especially because the performance improvement bounds of Section 4.3 are made very conservative by problem-dependent constants anyway. In the safe-RL literature, a similar false-Gaussianity assumption is done in [32], albeit with different motivations. The same authors call *semi-safe* an algorithm that makes a false, but reasonable assumption. Ours, however can be seen as an implementation choice, since Assumption 4.6 could be satisfied in more rigorous ways (see [5] for some examples).

Once the estimation error has been characterized, we can define the following lower-corrected and upper-corrected gradient norms:

$$\left\|\widehat{\nabla}^N_{\boldsymbol{\theta}}F\right\|^-_\infty = \left\|\widehat{\nabla}^N_{\boldsymbol{\theta}}F\right\|_\infty - \epsilon, \qquad \left\|\widehat{\nabla}^N_{\boldsymbol{\theta}}F\right\|^+_\infty = \left\|\widehat{\nabla}^N_{\boldsymbol{\theta}}F\right\|_\infty + \epsilon,$$

where $\epsilon = [\epsilon_1, \dots, \epsilon_m]^T$. To obtain high-probability variants of the Theorems from Section 4.3, it is enough to replace the true gradient norms with the corrected norms. The choice of lower-corrected versus upper-corrected must be made in order to preserve the inequalities. Note how (4.23) guarantees that $\left\|\nabla^N_{\boldsymbol{\theta}}F\right\|^-_\infty$ is always positive, allowing to safely take the square in inequalities. We now provide the high-probability variants of the Theorems from Section 4.3:

Theorem 4.7 (Adapted from Corollary 3.8 from [5]). *Assuming $\sigma = const$, update rule (4.11), where $\nabla_{\boldsymbol{v}}J$ is replaced by an estimator $\widehat{\nabla}_{\boldsymbol{v}}J$ satisfying Assumption 4.6, guarantees* ***with probability at least*** $(1-\delta)^m$*:*

$$J(\boldsymbol{v}^{t+1}) - J(\boldsymbol{v}^t) \geq \alpha\left(\left\|\nabla_{\boldsymbol{v}}J(\boldsymbol{v}^t)\right\|^-_\infty\right)^2 - \alpha^2 c\left(\left\|\nabla_{\boldsymbol{v}}J(\boldsymbol{v}^t)\right\|^+_\infty\right)^2, \tag{4.24}$$

where $c = \frac{RM_\phi^2}{(1-\gamma)^2\sigma^2}\left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)}\right)$ and $|\mathcal{A}|$ is the volume of the action space. Guaranteed improvement is maximised by using a step size $\alpha^* = \frac{1}{2c}$, yielding:

$$J(\boldsymbol{v}^{t+1}) - J(\boldsymbol{v}^t) \geq \frac{\left(\left\|\nabla_{\boldsymbol{\theta}}J(\boldsymbol{v}^t)\right\|^-_\infty\right)^4}{4c\left(\left\|\nabla_{\boldsymbol{\theta}}J(\boldsymbol{v}^t)\right\|^+_\infty\right)^2} := C^{\boldsymbol{v}}_{max}, \tag{4.25}$$

with probability at least $(1-\delta)^m$. Moreover, for any $C \leq C^{\boldsymbol{v}}_{max}$, the following constraints on the step-size α:

$$|\alpha - \alpha^*| \leq \lambda_{\boldsymbol{v}} \alpha^*, \tag{4.26}$$

where $\lambda_{\boldsymbol{v}} = \sqrt{1 - C/C^{\boldsymbol{v}}_{max}}$, are enough to guarantee $J(\boldsymbol{v}^{t+1}) - J(\boldsymbol{v}^t) \geq C$ with probability at least $(1-\delta)^m$.

The gradients w.r.t. w are scalar, but we keep the norm notation for the sake of conciseness:

Theorem 4.8. *Assuming $o(|\Delta w|^3)$ terms can be neglected, the w-update in (4.12), where $\nabla_w J$ is replaced by an estimator $\widehat{\nabla}_w J$ satisfying Assumption 4.6, guarantees, with probability at least $(1-\delta)$:*

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \beta \left(\left\| \widehat{\nabla}_w J(\boldsymbol{v}^{t+1}, w^t) \right\|^{-}_{\infty} \right)^2 - \beta^2 d \left(\left\| \widehat{\nabla}_w J(\boldsymbol{v}^{t+1}, w^t) \right\|^{+}_{\infty} \right)^2, \tag{4.27}$$

where $d = \frac{R}{(1-\gamma)^2} \left(\frac{\psi|\mathcal{A}|}{2\sigma_w} + \frac{\gamma}{1-\gamma} \right)$ and $\psi = 4(\sqrt{7}-2)\exp(\sqrt{7}/2-2)/\sqrt{2\pi} \simeq 0.524$. Guaranteed improvement is maximised by using a step size $\beta^* = \frac{1}{2d}$, yielding:

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \frac{\left(\left\| \widehat{\nabla}_w J(\boldsymbol{v}^{t+1}, w^t) \right\|^{-}_{\infty} \right)^4}{4d \left(\left\| \widehat{\nabla}_w J(\boldsymbol{v}^{t+1}, w^t) \right\|^{+}_{\infty} \right)^2} := C^{w}_{max}, \tag{4.28}$$

with probability at least $(1-\delta)^m$. Moreover, for any $\nu C \leq C^w_{max}$, the following constraints on the step-size β:

$$|\beta - \beta^*| \leq \lambda_w \beta^*, \tag{4.29}$$

where $\lambda_w = \sqrt{1 - \nu C / C^w_{max}}$, are enough to guarantee $J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \nu C$ with probability at least $(1-\delta)^m$.

Theorem 4.9. *Assuming $o(|\Delta w|^3)$ terms can be neglected, the w-update in (4.18), where $\nabla_w J$ and $\nabla_w \mathcal{L}$ are replaced by estimators $\widehat{\nabla}_w J$ and $\widehat{\nabla}_w \mathcal{L}$, respectively, satisfying Assumption 4.6, guarantees, for any (possibly negative) step size β, with probability at least $(1-\delta)^m$:*

$$\begin{aligned} J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \beta & \left\| \widehat{\nabla}_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t) \right\|^{-}_{\infty} \left\| \widehat{\nabla}_w J(\boldsymbol{v}^{t+1}, w^t) \right\|^{-}_{\infty} \\ & - \beta^2 d \left(\left\| \widehat{\nabla}_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t) \right\|^{+}_{\infty} \right)^2. \end{aligned}$$

Guaranteed improvement is maximised by using a step size
$\tilde{\beta} = \frac{\left\| \widehat{\nabla}_w J(\boldsymbol{v}^{t+1}, w^t) \right\|^{-}_{\infty} \left\| \widehat{\nabla}_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t) \right\|^{-}_{\infty}}{2d \left(\left\| \widehat{\nabla}_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t) \right\|^{+}_{\infty} \right)^2}$*, yielding:*

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \left(\frac{\left\| \widehat{\nabla}_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t) \right\|^{-}_{\infty} \left\| \widehat{\nabla}_w J(\boldsymbol{v}^{t+1}, w^t) \right\|^{-}_{\infty}}{2d \left\| \widehat{\nabla}_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^t) \right\|^{+}_{\infty}} \right)^2 := C^{w}_{max},$$

with probability at least $(1-\delta)^m$. *Moreover, for any* $\nu C \leq C^w_{max}$*, the following constraints on the step-size* β*:*

$$|\beta - \tilde{\beta}| \leq \lambda_w |\tilde{\beta}|, \tag{4.30}$$

where $\lambda_w = \sqrt{1 - {}^{\nu C}/_{C^w_{max}}}$*, are enough to guarantee* $J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^{t+1}, w^t) \geq \nu C$ *with probability at least* $(1-\delta)^m$.

These results offer high-probability bounds on the performance improvement, that can be used to further customise the safety constraint needs for different values of δ and ϵ.

Chapter 5

Experimental analysis

In Chapter 4 we have presented a new general framework for safe reinforcement learning that can be used to design safe constraints depending on the user needs. In particular we have divided the safety constraints in two families, that we called type-I and type-II.
Type-I constraints target more critical situations, where we need a strict safety guarantee. Type-II constraints, instead, target a more flexible situation, where we are only concerned on guarantees over a learning iteration. Within this framework, we identified several variants, that fulfil different needs in practical situations. Specifically, in Section 4.4, we have described the Safely-Exploring Policy Gradient (SEPG) algorithm and how to adapt it to implement such variants.

In this chapter, we will evaluate all these variants on simulated continuous control tasks: in Section 5.1, we will consider the linear-quadratic Gaussian (LQG) control problem. This is a well-known control task for which a closed-form solution exists, which let us study in more details the behaviour of our algorithms. Then, in Section 5.2, we will consider the Mountain Car problem, which is a complex task that requires a particularly careful exploration strategy.

5.1 Linear Quadratic Gaussian Controller

The linear-quadratic Gaussian (LQG) regulator problem [1] is a continuous MDP defined by a transition model:

$$s_{t+1} \sim \mathcal{N}(s_t + a_t, \eta_0^2),$$

where the next state is obtained by summing the action and some noise to the current state, a Gaussian policy:

$$a_t \sim \mathcal{N}(\boldsymbol{v}^\intercal s_t, e^{2w}),$$

and a reward function that depends on the distance to the goal and the magnitude of the action:

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

Both action and state space are continuous and bounded to the interval $\mathcal{S}, \mathcal{A} = [-2, 2]$ and the initial state s_0 is drawn uniformly at random from $\mathcal{S}$. The goal of this task is to bring the state s_t to zero.
The main difficulty of this task is reflected in the reward function, that depends both on the state and the action. This poses a trade-off on the magnitude of the action: we want to reach the goal fast enough to avoid the penalization given by the distance to the goal, but slow enough to avoid a strong penalization on the action.

This problem admits a closed-form solution for Gaussian policies with mean linear in the state space. In all our experiments, we use a discount factor of $\gamma = 0.99$, which gives an optimal parameter $\boldsymbol{v}^* \approx -0.615$ corresponding to an expected deterministic performance $J(\boldsymbol{v}^*, -\infty) \approx -7.753$. We always start from an initial policy with $\boldsymbol{v}^0 = -0.1$. This is a rather good starting point, since it is far from the optimum and values closer to 0 make the controller less stable.
The knowledge of the optimal policy $\boldsymbol{\theta}^*$ gives us the possibility to compare the algorithms on how close they are w.r.t. the optimal solution.

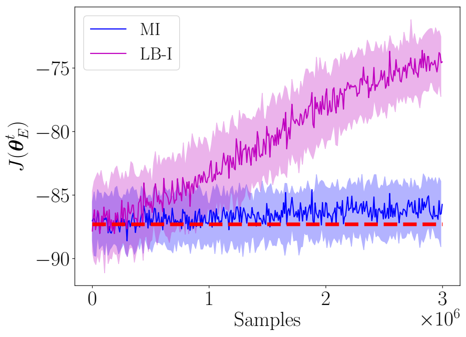
We will gradually present the results in this way: we will compare Monotonic Improvement (MI) with Lower Bounded I (LB-I) first, which are the two implementations of type-I safety. Next we will also add the other variants and we will make some considerations about the difference between type-I and type-II safety. Finally, we will compare Zero Variance (ZV) with some alternative implementations that simultaneously perform a $\boldsymbol{v}$-update and a w-update, thus requiring less trajectories, but providing no guarantees.

5.1.1 Type-I safety constraints

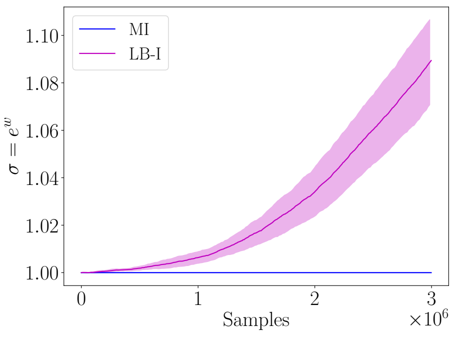
Here we want to focus on the comparison between MI and LB-I. They both guarantee a type-I safety constraint of the kind:

$$J(\boldsymbol{\theta}^{t+1}) \geq J_B^t$$

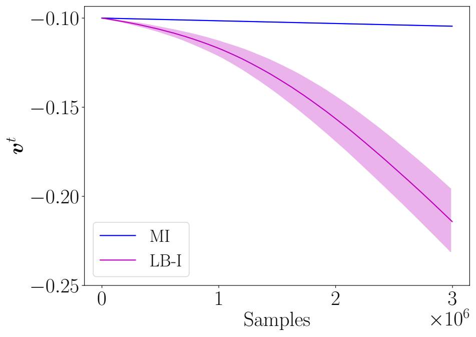
for each update. The main difference is the value of the baseline: MI sets the baseline to be the performance of the policy at the previous iteration, namely $J_B^t = J(\boldsymbol{\theta}^t)$, while LB-I lower-bounds the performance of the exploratory policy w.r.t. the initial policy, that is, $J_B^t = J(\boldsymbol{\theta}^0)$, that guarantees $J(\boldsymbol{\theta}^t) \geq J(\boldsymbol{\theta}^0)$ for each update.
This adds a new degree of freedom that greatly increase the learning speed, as shown in Figure 5.1. In particular, the advantage of LB-I is that it is not required to improve the solution on every step, but it can afford to lose up to $J(\boldsymbol{\theta}^t) - J(\boldsymbol{\theta}^0)$, that is, the advantage of the current policy w.r.t. the initial policy. This difference can be seen after some iterations of the algorithm, where LB-I's advantage over MI increases, allowing it to take even larger updates.



(a) Performance of the exploratory policy Note that: $\boldsymbol{\theta}_E^t = \boldsymbol{\theta}_T^t$



(b) Evolution of the variance parameter σ



(c) Evolution of the Gaussian mean parameter $\boldsymbol{v}^t$

Figure 5.1: Comparison between MI and LB-I implementations. The less stringent guarantees given by LB-I allows for a faster convergence speed of the mean parameter $\boldsymbol{v}^t$ and an increase in exploration.

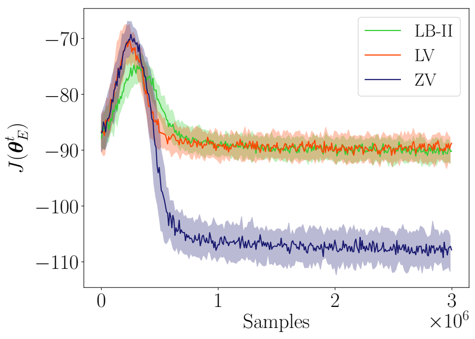
Moreover, LB-I safely increases the exploration parameter σ (Figure 5.1b) by following the newly introduced quantity $\nabla_w \mathcal{L}$, that further increases the convergence speed of the parameter $\boldsymbol{v}^t$, as shown in Figure 5.1c. The MI variant, instead, is not able to significantly change the variance σ because of its high conservativeness.

5.1.2 Type-II safety constraints

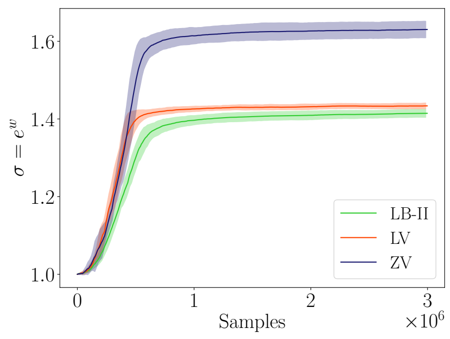
In this section, we will compare the behaviour of all type-II variants of SEPG, shown in Figure 5.2.
Specifically, (a) shows the performance of the exploratory policies; (b) the evolution of σ in the different variants; (c) the evolution of the mean parameter $\boldsymbol{v}^t$; (d) the safety guarantee that is implemented; (e) the deterministic policy optimized by LV and ZV; (f) the evolution of the budget B^t.
We identify the following interesting behaviours:

- Contrary to type-I variants (MI, LB-I), that tend to directly improve $J(\boldsymbol{\theta})$, type-II variants (LB-II, LV, ZV) invest the performance improvements for later usage. Specifically, a positive performance improvement generates budget, that will be spent in exploration. The budget constitutes an upper-bound for the $\boldsymbol{v}$-updates and w-updates, hence a high budget have the effect of decreasing the conservativeness of the algorithm by allowing bounded performance loss.
- In Figure 5.2d we can see that the type-II safety guarantee:
$$\frac{1}{K}\sum_{k}\left(J(\boldsymbol{\theta}^{k(t)}) + J(\boldsymbol{\theta}_T^{k(t)})\right)/2 \geq J_B^t$$
is satisfied. We have highlighted the performance of the baseline policy with a red dashed line.
In particular note that LB-I, LV and ZV belong to different scales. This is due to the magnitude of the target policy: for LB-II, we do not evaluate the target policy (since we set $\boldsymbol{\theta}_T^t = \boldsymbol{\theta}_E^t$); for LV, we evaluate a target policy that is less stochastic than the exploratory policy, but it is still close to the exploratory policy; for ZV, we evaluate the deterministic policy, that has a higher performance than the low-variance policy. This can be seen in Figure 5.2e.
Moreover, we can see that they both end slightly below the baseline. This is due to two factors: the first is the gradient approximation, that adds some errors on our estimations. The second factor is related to the behaviour of the algorithm: as it is designed, the algorithm will try to spend all the available budget in bounded-worsening updates. This is fine as long as new income is added to the budget. When the parameter is close to the optimal value, though, it becomes hard to increase the budget. As it happens, the budget becomes slightly negative and the algorithm is not capable of restoring a positive value for it.
- The evolution of the online performance measure $J(\boldsymbol{\theta}_E)$ is of particular interest: it grows to a high value at the beginning and then drops. This behaviour can be explained by looking at the evolution of σ and $\boldsymbol{v}^t$: as $\boldsymbol{v}$ converges toward the optimal value, the performance of the policy increases and some of the gained budget is spent in raising σ. However, when $\boldsymbol{v}$ is close to the optimum, there are no big improvements reflected in the performance measure, hence the budget is spent solely on exploration. This increases the exploration costs and explains the drop in the policy performance $J(\boldsymbol{\theta}_E^t)$. Note that for ZV the drop is also related with the gain in the target policy.
- The evaluation of the target policy for LV and ZV is shown in Figure 5.2e. The performance of the deterministic policy optimized by ZV ends very close to the optimum $J(\boldsymbol{v}^*, -\infty) \approx -7.753$. The low-variance target policy optimized by LV, instead, is very similar to the exploratory policy $\boldsymbol{\theta}_E^t$. This is because the bounds used to compute the low-variance policy do not allow for a significant change of the variance, which remains only slightly reduced.

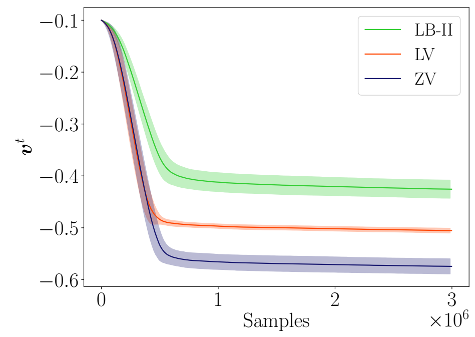
In general we can see that ZV converges faster to the optimal value $\boldsymbol{v}^*$ than the other type-II variants. The main reason is that the evaluation of the deterministic



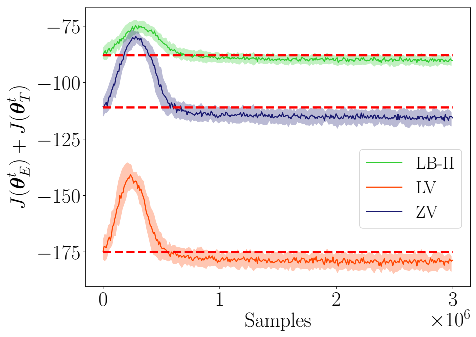
(a) Performance of the exploratory policy



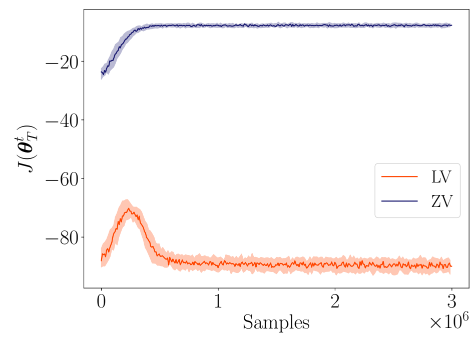
(b) Increase in the variance parameter σ



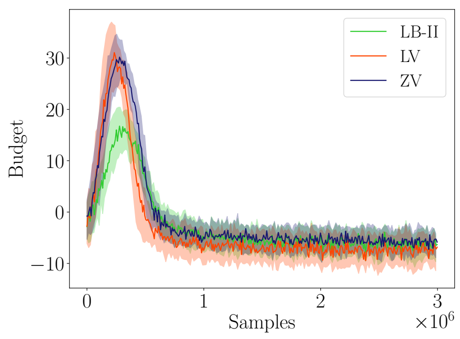
(c) Convergence speed of the Gaussian mean parameter $\boldsymbol{v}^t$



(d) The type-I and type-II safety guarantee. For MI,LB-I and LB-II we have $\boldsymbol{\theta}_T^t = \boldsymbol{\theta}_E^t$



(e) Performance of the target policy for LV and ZV



(f) Evolution of the budget

Figure 5.2: Comparison between SEPG type-II variants presented in Section 4.4. Depending on the user needs, the user can choose between different safe approaches.

policy increases the amount of available budget. The same behaviour can be seen for LV, but, since the low-variance policy still remain very close to the stochastic one, the improvement is much less than for ZV.
All these variants are notably faster than the type-I implementations described before.

	Adam			$\alpha = const$							
σ	1e-4	1e-05	1e-06	1e-4	1e-5	1e-6	MI	LB-I	LB-II	LV	ZV
0.1	0.126	0.432	0.506	0.132	**0.026**	0.228	0.515	0 .510	0.368	0.429	0.435
0.2	0.08	0.429	0.506	0.124	**0.018**	0.201	0.515	0.494	0.039	0.028	0.056
0.5	0.061	0.428	0.506	0.122	0.012	0.125	0.511	0.29	0.009	0.01	**0.006**
1.0	0.046	0.422	0.505	0.122	0.011	0.069	0.497	**0.01**	0.021	0.021	0.028

Table 5.1: Upper bound confidence intervals at 95% of $|\boldsymbol{v} - \boldsymbol{v}^*|$ after 10,000 iterations.

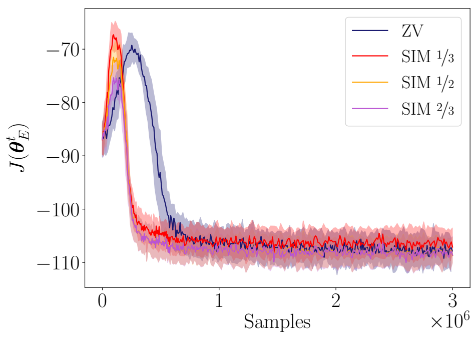
Next, we compare the convergence speed of these variants with other state-of-the-art non-safe approaches. Table 5.1 shows a 95%-confidence upper bound on the absolute error $|\boldsymbol{v} - \boldsymbol{v}^*|$ after 10,000 iterations for different initial values of σ. We compare our algorithms to REINFORCE with constant step size ($\alpha = \beta = const$) and Adam [40], an algorithm that starts from an initial learning rate, and subsequently adapts to the state of the optimization.
Our algorithms, of course, cannot outperform all possible combinations of manually-chosen step sizes, but we can see that our algorithms give comparable convergence speed without the need of tuning the step-size. This is an important aspect that allows SEPG to be more domain-independent.

In particular, we note that our algorithms perform better with higher initial values of σ. A possible explanation for this behaviour is that the bounds used to compute the optimal step size $\overline{\alpha}$ and $\overline{\beta}$ highly depend on σ: higher values of σ make the bounds less tight, thus speeding up convergence speed.
Moreover, we can see that the Monotonic Improvement (MI) case is in general the slowest to converge due to the more conservative behaviour of the algorithm. This is the main reason that led the investigation of other types of safety constraints that better fit the user needs.

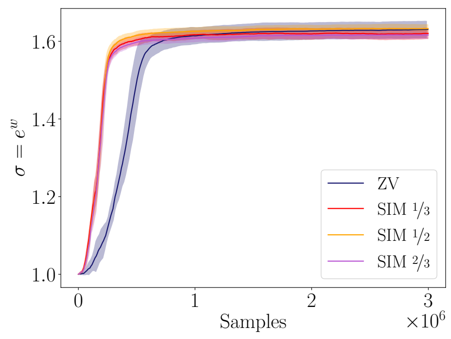
5.1.3 Alternative comparisons

As we have introduced in the end of Section 4.4, a possible way to further speed up the convergence speed of our algorithms is to reuse the collected trajectories and simultaneously perform a $\boldsymbol{v}$-update and a w-update. This comes at the cost of not having strict guarantees that the algorithm will satisfy the safety constraint. We tested this variant, called SEPG-SIM, in the same LQG setting.

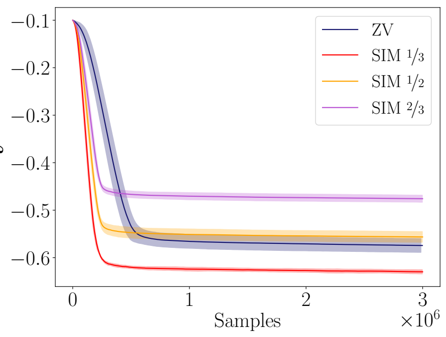
A problem that naturally arises when considering simultaneous updates is the allocation of the budget, that is, how to split the budget between the $\boldsymbol{v}$-update and the w-update. We did not study this allocation issue in-depth, but we have only considered a fixed-strategy: at every iteration, the budget B is split so that νB is used for the $\boldsymbol{v}$-update, and $(1-\nu)B$ is used for the w-update. Specifically, we tested values for $\nu \in \{1/3, 1/2, 2/3\}$.
The results are shown in Figure 5.3: as expected, the simultaneous approach required less samples to converge. Moreover, the safety constraint is still satisfied



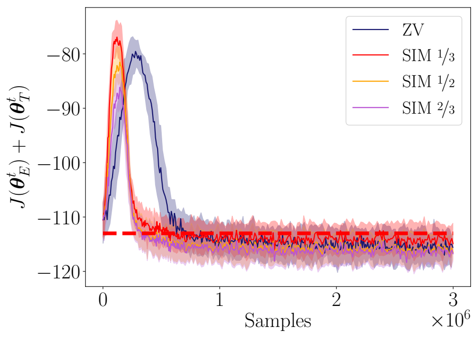
(a) Performance of the exploratory policy



(b) Evolution of the variance parameter σ



(c) Evolution of the Gaussian mean parameter $\boldsymbol{v}^t$



(d) The type-II safety guarantee.

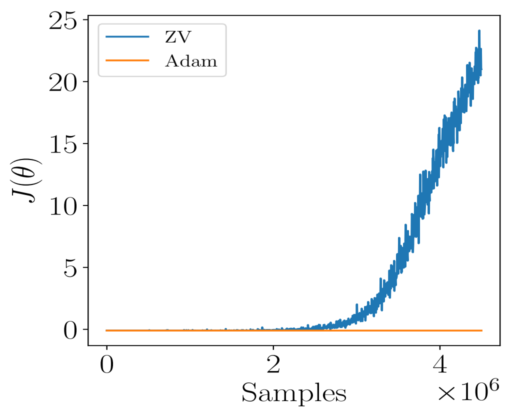
Figure 5.3: Comparison between ZV and an alternative variant of SEPG (SIM-ν) with simultaneous updates. The budget is split in this way: νB for the $\boldsymbol{v}$-update and $(1-\nu)B$ for the w-update.

despite this approximation. Finally, we can see the difference between the different budget allocation strategies in Figure 5.3c: allocating more budget to the w-update (SIM-1/3) gives a faster convergence for the parameter $\boldsymbol{v}^t$. This aspect shows the strict relation between exploration and exploitation, where the former influences the latter in a non-trivial way.

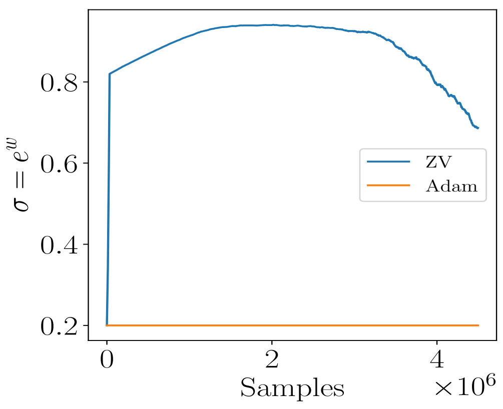
5.2 Mountain Car

We also tested our algorithms on the Mountain Car problem (in the *openai/gym* implementation) to show the advantages of our algorithm in a situation where exploration is particularly important.

We have introduced the problem of Mountain Car in Example 2.2. Briefly, a car is placed between two hills and the goal is to reach the top of the right hill by moving the car left or right. This problem is difficult for its exploration need, because the engine of the car is not powerful enough, hence the solution requires the agent to gain



(a) Performance of Adam and ZV algorithm.



(b) Evolution of the exploration parameter σ

Figure 5.4: Mountain Car task, starting from a very low variance

momentum by first going on the opposite direction. Moreover, the reward function includes a penalization on the magnitude of the action. This is where the problems begin: if the agent does not reach the goal soon enough, this penalization will push the agent to not move at all. This is a local maximum because the agent will get zero reward by standing still, while it will receive a negative reward if it moves but does not reach the goal. This penalty clearly outlines the role of exploration in a reinforcement learning task: we have to take actions that possibly yield bad immediate rewards, in order to obtain a good benefit in the future. This task, in particular, requires the agent to reach the goal at least once, before actually solving the problem.

We set $\sigma^0 = 0.2$, which is not enough to reach the goal. By providing a nominal initial budget ($B^0 = 1$), our method SEPG-ZV starts investing more and more in exploration until it reaches the goal, and is ultimately able to solve the problem. An important role is played by the choice of following the gradient $\nabla_w \mathcal{L}(\boldsymbol{v}^t, w^t)$ instead of the naïve gradient $\nabla_w J(\boldsymbol{v}^t, w^t)$.
Figure 5.4 shows the performance (on the left) and the policy variance (on the right) in our experiment. As we can see, a naïve gradient ascent update (Adam in the picture, but the same can be shown for any reasonable fixed step size), is not far-sighted enough to see any advantage in increasing σ, and gets stuck immediately.

Chapter 6

Conclusions

In this thesis, we have introduced a new framework and new techniques for safe reinforcement learning that combines some of the results done in this field. We identified two families of safety constraints: type-I safety, for the most critical situations where we must have guarantees on every policy update; type-II safety, for less critical situations where the safety constraint can be guaranteed over a *learning iteration*. Within these families, we have also characterized several situations that correspond to practical user needs, providing a label for each one of them: **Monotonic Improvement** (MI), that gives the strictest guarantee and can be used in critical applications or when costs/risks are not modelled in the reward; **Lower Bounded I** (LB-I), that sets the performance of the initial policy as a lower bound for each subsequent update, and can be used when we already have a good initial policy but we want to improve it; **Lower Bounded II** (LB-II), that guarantees type-II safety over a learning iteration and can be used in an on-line scenario where we do not want to perform worse than the initial policy on average; **Low-Variance** (LV) that adds the possibility to test the quality of a target policy under low-variance conditions (e.g., a prototype); **Zero Variance** (ZV) where we sacrifice some guarantees and we test a deterministic controller, which gives a more realistic evaluation of the target policy and possibly allows for a faster convergence speed.

We then extended the results on performance improvement bounds from [4, 5] on policy gradient with Gaussian policies to include an exploratory behaviour. We introduced a new quantity $\nabla_w \mathcal{L}(\boldsymbol{v}^t, w^t)$ that was used as an alternative approach of $\nabla_w J(\boldsymbol{v}^t, w^t)$ to handle exploration. This new approach was able to tackle a difficult problem such as Mountain Car.

Finally, we provided Safely-Exploring Policy Gradient (SEPG), a general algorithm that can be customized to every safety constraint introduced so far. The main idea is to give to the user the safety requirements he/she needs, and no more. We evaluated these variants on continuous control tasks such as the Linear Quadratic Gaussian control problem. Experiments showed that the algorithm effectively guarantees the imposed safety constraint and simultaneously allows for a more exploratory behaviour than naïve approaches. More importantly, these algorithms adapt to the environment and represent a step towards policy gradient algorithms that can be employed efficiently and safely in real applications, without requiring a strong domain knowledge nor a long period of hyper-parameter tuning.

Future works can focus on extending the theoretical guarantees to other classes of policies (e.g., softmax for discrete MDPs). This would increase the range of problems where safe reinforcement learning can be applied. Another interesting approach would be to develop new methods to automatically select other meta-parameters in SEPG, such as the batch size (as done in [5]), or the budget-allocation strategy. The former can be easily employed with little effort. The latter requires more thought, but can yield interesting results because a different strategy can drastically change the behaviour of the algorithm. Moreover, a better strategy can solve the 'large budget' problem. This problem is as follows: when we raise the budget through our improvements in the policy performance, we are likely to spend it in exploration. However, when this is not possible, or have marginal effects, we remain with a high positive budget. While this can seem a good thing, it can cause problems, since we will make large bounded-worsening updates that destabilize the algorithm and are likely to incur in overshooting. Two possible ways of tackling this problem are: i) discount the budget with a constant factor γ. This, however, introduces a new hyper-parameter that is domain dependent. ii) reset the budget to 0 (or a constant value) 'once in a while'. The meaning behind this approach is to make a checkpoint of the current policy performance. In fact, by resetting the budget, we are basically restarting the algorithm and setting the initial policy to be the current one. Choosing when to make a checkpoint can be a non-trivial task: too many checkpoints can drastically reduce convergence speed, while making too few can still suffer from the 'high budget' problem.

Finally, another possible extension to this work can be to study new exploration techniques that safely explore the environment. For example, we can reduce the amount of exploration when we think we have collected enough samples from the environment, increasing the performance in an online scenario. A different approach would be to adapt exploration for each state: we can increase the exploration of states we visited less frequently, while we can exploit the states we visited more often. However, as we have seen, the exploration-exploitation dilemma is a really challenging task. There is not a general solution for this problem yet, so we need to keep pushing in this direction and devise better solutions.

Bibliography

[1] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, May 2008.

[2] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *IN PROC. 19TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*, pages 267–274, 2002.

[3] Matteo Pirotta, Marcello Restelli, Alessio Pecorino, and Daniele Calandriello. Safe policy iteration. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 307–315, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[4] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Adaptive step-size for policy gradient methods. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1394–1402. Curran Associates, Inc., 2013.

[5] Matteo Papini, Matteo Pirotta, and Marcello Restelli. Adaptive batch size for safe policy gradients. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3591–3600. Curran Associates, Inc., 2017.

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[7] Richard Bellman. Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228 – 239, 1958.

[8] R. A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, MA, 1960.

[9] Martin L. Puterman and Shelby L. Brumelle. On the convergence of policy iteration in stationary dynamic programming. *Mathematics of Operations Research*, 4(1):60–69, 1979.

[10] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, Nov 1994.

[11] John N. Tsitsiklis. On the convergence of optimistic policy iteration. *J. Mach. Learn. Res.*, 3:59–72, March 2003.

[12] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988.

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

[14] G A. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. 11 1994.

[15] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[16] P. Dyer and S.R. McReynolds. *The Computation and Theory of Optimal Control*. Mathematics in Science and Engineering : a series of monographs and textbooks. Academic Press, 1970.

[17] L. Hasdorff. *Gradient optimization and nonlinear control*. John Wiley & Sons Australia, Limited, 1976.

[18] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, 23(3):462–466, 09 1952.

[19] Peter W. Glynn. Likelihood ratio gradient estimation for stochastic systems. *Commun. ACM*, 33(10):75–84, October 1990.

[20] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256, 1992.

[21] Peter L. Bartlett and Jonathan Baxter. Infinite-horizon policy-gradient estimation. *CoRR*, abs/1106.0665, 2011.

[22] L. Rade, P. Vachenauer, and B. Westergren. *Springers mathematische Formeln: Taschenbuch für Ingenieure, Naturwissenschaftler, Informatiker, Wirtschaftswissenschaftler*. Springer Berlin Heidelberg, 2000.

[23] Sham Kakade. A natural policy gradient. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pages 1531–1538. MIT Press, 2001.

[24] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural Comput.*, 10(2):251–276, February 1998.

[25] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180 – 1190, 2008. Progress in Modeling, Theory, and Application of Computational Intelligenc.

[26] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust Region Policy Optimization. *ArXiv e-prints*, February 2015.

[27] Dimitri P. Bertsekas. Approximate policy iteration: a survey and some new methods. *Journal of Control Theory and Applications*, 9(3):310–335, Aug 2011.

[28] Paul Wagner. A reinterpretation of the policy oscillation phenomenon in approximate policy iteration. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2573–2581. Curran Associates, Inc., 2011.

[29] Matteo Pirotta and Marcello Restelli. Cost-sensitive approach to batch size adaptation for gradient descent. *CoRR*, abs/1712.03428, 2017.

[30] Julie Nutini, Mark Schmidt, Issam Laradji, Michael Friedlander, and Hoyt Koepke. Coordinate descent converges faster with the gauss-southwell rule than random selection. In *International Conference on Machine Learning*, pages 1632–1641, 2015.

[31] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup. Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. *ArXiv e-prints*, August 2017.

[32] Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence policy improvement. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2380–2388, Lille, France, 07–09 Jul 2015. PMLR.

[33] Doina Precup, Richard S. Sutton, and Satinder P. Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 759–766, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[34] Marek Petrik, Mohammad Ghavamzadeh, and Yinlam Chow. Safe policy improvement by minimizing robust baseline regret. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2306–2314, USA, 2016. Curran Associates Inc.

[35] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement Learning with Deep Energy-Based Policies. *arXiv:1702.08165 [cs]*, February 2017. arXiv: 1702.08165.

[36] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum Entropy Inverse Reinforcement Learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[37] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[38] Clement Gehring and Doina Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1037–1044. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[39] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[40] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[41] Mark S Pinsker. Information and information stability of random variables and processes. 1960.

[42] Manuel Gil, Fady Alajaji, and Tamas Linder. Rényi divergence measures for commonly used univariate continuous distributions. *Information Sciences*, 249:124–131, 2013.

Appendix A

Proofs

In this section we provide formal proofs of all the formal statements made in this work.

Theorem 4.1. *Any iterative learning algorithm that maintains $B^t \geq 0$ fulfills type-II safety with respect to a constant baseline $J_B = [J(\boldsymbol{\theta}_E^0) + J(\boldsymbol{\theta}_T^0)]/2$.*

Proof. By telescoping the sum in the definition of budget:

$$\begin{aligned} B^t &= \sum_{k=0}^{t} J(\boldsymbol{\theta}^t) - J(\boldsymbol{\theta}^{t-1}) + J(\boldsymbol{\theta}^t_{target}) - J(\boldsymbol{\theta}^{t-1}_{target}) = \\ &= J(\boldsymbol{\theta}^t) - J(\boldsymbol{\theta}^0) + J_{target}(\boldsymbol{\theta}^t) - J_{target}(\boldsymbol{\theta}^0). \end{aligned}$$

By rearranging terms, $B^t \geq 0$ implies:

$$J(\boldsymbol{\theta}^t) + J_{target}(\boldsymbol{\theta}^t) \geq J(\boldsymbol{\theta}^0) + J_{target}(\boldsymbol{\theta}^0).$$

$\square$

Theorem 4.2 (From Theorem 3.3 in [5])**.** *Assuming $w^t = const$ for all t, update rule (4.11) guarantees:*

$$J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t) \geq \alpha \left\|\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t)\right\|_\infty^2 - \alpha^2 c \left\|\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t)\right\|_\infty^2, \tag{4.8}$$

where $c = \frac{RM_\phi^2}{(1-\gamma)^2\sigma^2}\left(\frac{|\mathcal{A}|}{\sqrt{2\pi}\sigma} + \frac{\gamma}{2(1-\gamma)}\right)$ and $|\mathcal{A}|$ is the volume of the action space. Guaranteed improvement is maximised by using a step size $\alpha^* = \frac{1}{2c}$, yielding:

$$J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t) \geq \frac{\left\|\nabla_{\boldsymbol{\theta}} J(\boldsymbol{v}^t, w^t)\right\|_\infty^2}{4c} := C_{max}^{\boldsymbol{v}}. \tag{4.9}$$

Moreover, for any $C \leq C_{max}^{\boldsymbol{v}}$, the following constraints on the step-size α:

$$|\alpha - \alpha^*| \leq \lambda_{\boldsymbol{v}} \alpha^*, \tag{4.10}$$

where $\lambda_{\boldsymbol{v}} = \sqrt{1 - C/C_{max}^{\boldsymbol{v}}}$, are enough to guarantee $J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t) \geq C$.

Proof. The first statement (4.8) is adapted from Theorem 3.3 in [5] by setting $\boldsymbol{\theta} = \boldsymbol{v}$ and:

$$\Lambda_{kk} = \alpha \quad \text{if} \quad k = \arg\max_i |\nabla_{\boldsymbol{v}_i} J(\boldsymbol{v})| \quad \text{else} \quad 0.$$

The second statement (4.9) is exactly Corollary 3.4 from [5]. The third statement (4.10) derives from (4.8) by requiring:

$$\alpha \left\|\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t)\right\|_\infty^2 - \alpha^2 c \left\|\nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t)\right\|_\infty^2 \geq C,$$

and solving the inequality for α. Note that there is no real solution for $C > C_{\max}^{\boldsymbol{v}}$. □

Before proving Theorem 4.3, we need the following lemmas:

Lemma A.1. *The second derivative w.r.t. the standard-deviation parameter w of a policy parametrized as in Equation (4.7) is bounded as follows:*

$$\left|\frac{\partial^2 \pi_{\boldsymbol{\theta}}(a|s)}{\partial w^2}\right| \leq \frac{\psi}{\sigma_w},$$

where $\psi = \frac{4(\sqrt{7}-2)e^{\frac{\sqrt{7}}{2}-2}}{\sqrt{2\pi}}$.

Proof. Let $\chi = \left(\frac{a-\mu_{\boldsymbol{v}}}{\sigma_w}\right)^2$. A first useful fact is:

$$\frac{\partial \chi}{\partial w} = (a-\mu_{\boldsymbol{v}})^2 \frac{\partial \chi}{\partial w} e^{-2w} = -2\chi. \tag{A.1}$$

Writing the policy as:

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{1}{\sqrt{2\pi}\sigma_w} e^{-\chi/2},$$

derivatives w.r.t. w can be easily computed as:

$$\frac{\partial \pi_{\boldsymbol{\theta}}(a|s)}{\partial w} = -\pi_{\boldsymbol{\theta}}(a|s) + \pi_{\boldsymbol{\theta}}(a|s)(-\frac{1}{2})(-2\chi) = (\chi-1)\pi_{\boldsymbol{\theta}}(a|s), \tag{A.2}$$

$$\frac{\partial^2 \pi_{\boldsymbol{\theta}}(a|s)}{\partial w^2} = -2\chi\pi_{\boldsymbol{\theta}}(a|s) + (\chi-1)(\chi-1)\pi_{\boldsymbol{\theta}}(a|s) = (\chi^2-4\chi+1)\pi_{\boldsymbol{\theta}}(a|s). \tag{A.3}$$

Next, we study the continuous function:

$$f(\chi) := (\chi^2 - 4\chi + 1)e^{-\frac{\chi}{2}} = \sqrt{2\pi}\sigma_w \frac{\partial^2 \pi_{\boldsymbol{\theta}}(a|s)}{\partial w^2},$$

constrained, of course, to $\chi \geq 0$. We find that $f(\chi)$ has two stationary points:

- $\chi_1 = 4 + \sqrt{7}$, with $f(\chi_1) = 4(\sqrt{7}+2)e^{-\frac{\sqrt{7}}{2}-2} \simeq 0.67$;
- $\chi_2 = 4 - \sqrt{7}$, with $f(\chi_2) = -4(\sqrt{7}-2)e^{\frac{\sqrt{7}}{2}-2} \simeq -1.31$, which is also the global minimum.

Moreover, $f(0) = 1$ and $f(\chi) \to 0$ as $\chi \to +\infty$, so $\chi_0 = 0$ is the global maximum in the region of interest. We can then state that $|f(\chi)| \leq |f(\chi_2)| = 4(\sqrt{7}-2)e^{\frac{\sqrt{7}}{2}-2}$, the maximum absolute value that $f(\chi)$ takes in $[0, +\infty]$. Finally:

$$\left|\frac{\partial^2 \pi_{\boldsymbol{\theta}}(a|s)}{\partial w^2}\right| = \frac{|f(\chi)|}{\sqrt{2\pi}\sigma_w} \leq \frac{4(\sqrt{7}-2)e^{\frac{\sqrt{7}}{2}-2}}{\sqrt{2\pi}\sigma_w} := \frac{\psi}{\sigma_w}.$$

□

Lemma A.2. *Let $w' = w + \Delta w$. Then, the difference of the corresponding policies can be bounded as follows:*

$$\pi_{\boldsymbol{v},w'}(a|s) - \pi_{\boldsymbol{v},w}(a|s) \geq \nabla_w \pi_{\boldsymbol{\theta}}(a|s)\Delta w - \frac{\psi \Delta w^2}{2\sigma_w},$$

where ψ is defined as in Lemma A.1.

Proof. The Taylor expansion of $\pi_{\boldsymbol{v},w'}$ is:

$$\pi_{\boldsymbol{v},w'}(a|s) = \pi_{\boldsymbol{v},w}(a|s) + \nabla_w \pi_{\boldsymbol{\theta}}(a|s)\Delta w + R_1(\Delta w), \tag{A.4}$$

where $R_1(\Delta w)$ is the remainder of the series, given by:

$$R_1(\Delta w) = \left.\frac{\partial^2 \pi_{\boldsymbol{\theta}}(a|s)}{\partial w^2}\right|_{w+\epsilon\Delta w} \frac{\Delta w^2}{2} \qquad \text{for some } \epsilon \in (0,1).$$

We can bound the remainder using Lemma A.1:

$$R_1(\Delta w) \geq -\sup_{\epsilon} \left| \left.\frac{\partial^2 \pi_{\boldsymbol{\theta}}(a|s)}{\partial w^2}\right|_{w+\epsilon\Delta w} \frac{\Delta w^2}{2} \right| \geq -\frac{\psi \Delta w^2}{2\sigma_w}, \tag{A.5}$$

where the last inequality is from Lemma A.1. The lemma follows from (A.4) and (A.5) by rearranging terms. □

Lemma A.3. *Let $w' = w + \Delta w$. Then, by neglecting $o(\Delta w^3)$ terms, the squared Chebyshev distance among the corresponding policies can be bounded as follows:*

$$\left\|\pi_{\boldsymbol{v},w'} - \pi_{\boldsymbol{v},w}\right\|_\infty^2 \leq 2\Delta w^2.$$

Proof. We have that:

$$\begin{aligned}
\left\|\pi_{\boldsymbol{v},w'} - \pi_{\boldsymbol{v},w}\right\|_\infty^2 &\leq \sup_{s\in\mathcal{S}} \left\|\pi_{\boldsymbol{v},w'}(\cdot|s) - \pi_{\boldsymbol{v},w}(\cdot|s)\right\|_1^2 \\
&\leq \sup_{s\in\mathcal{S}} \left(2H(\pi_{\boldsymbol{v},w'}(\cdot|s) \| \pi_{\boldsymbol{v},w}(\cdot|s))\right) &\text{(A.6)} \\
&= 2\left(\log\frac{\sigma_w}{\sigma_{w'}} + \frac{\sigma_{w'}^2}{2\sigma_w^2 - \frac{1}{2}}\right) &\text{(A.7)} \\
&= -2\Delta w + e^{2\Delta w} - 1 &\text{(A.8)} \\
&= 2\Delta w^2 + o(\Delta w^3),
\end{aligned}$$

where $H(\cdot\|\cdot)$ is the Kullback-Leibler (KL) divergence, (A.6) is from Pinsker's inequality [41], (A.7) is from the expression of the KL divergence for Gaussian distributions with equal mean [42], and (A.8) is from expanding $e^{\Delta w}$ in Taylor's series up to the second order. Note how the sup on states can be dropped since σ is state-independent. $\square$

Theorem 4.3. *Assuming $o(|\Delta w|^3)$ terms can be neglected,*[1] *the w-update in (4.12) guarantees:*

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \beta \nabla_w J(\boldsymbol{v}^{t+1}, w^t)^2 - \beta^2 d \nabla_w J(\boldsymbol{v}^{t+1}, w^t)^2, \tag{4.15}$$

where $d = \frac{R}{(1-\gamma)^2}\left(\frac{\psi|\mathcal{A}|}{2\sigma_w} + \frac{\gamma}{1-\gamma}\right)$ and $\psi = {}^{4(\sqrt{7}-2)\exp(\sqrt{7}/2-2)}/\sqrt{2\pi} \simeq 0.524$. Guaranteed improvement is maximised by using a step size $\beta^* = \frac{1}{2d}$, yielding:

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \frac{\nabla_w J(\boldsymbol{v}^{t+1}, w^t)^2}{4d} := C^w_{max}. \tag{4.16}$$

Moreover, for any $\nu C \leq C^w_{max}$, the following constraints on the step-size β:

$$|\beta - \beta^*| \leq \lambda_w \beta^*, \tag{4.17}$$

where $\lambda_w = \sqrt{1 - {}^{\nu C}/C^w_{max}}$, are enough to guarantee $J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^t) \geq \nu C$.

Proof. We proof the three statements in order:
Statement (4.15): we start from Lemma 3.1 from [4]:

$$\begin{aligned} J(\boldsymbol{\theta}') - J(\boldsymbol{\theta}) \geq & \frac{1}{1-\gamma}\int_{\mathcal{S}} d^{\boldsymbol{\theta}}_{\rho}(s) \int_{\mathcal{A}} (\pi_{\boldsymbol{\theta}'}(a|s) - \pi_{\boldsymbol{\theta}}(a|s)) Q^{\boldsymbol{\theta}}(s,a)\,\mathrm{d}a\,\mathrm{d}s \\ & - \frac{\gamma}{2(1-\gamma)^2} \|\pi_{\boldsymbol{\theta}'} - \pi_{\boldsymbol{\theta}}\|^2_{\infty} \left\|Q^{\boldsymbol{\theta}}\right\|_{\infty}, \end{aligned} \tag{A.9}$$

where $d^{\boldsymbol{\theta}}_{\rho}(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \Pr(s_t = s|\pi_{\boldsymbol{\theta}}, \rho)$ is the discounted future-state distribution and $Q^{\boldsymbol{\theta}}(s,a) = \mathcal{R}(s,a) + \gamma\int_{\mathcal{S}}\mathcal{P}(s'|s,a)\int_{\mathcal{A}}\pi_{\boldsymbol{\theta}}(a|s)Q^{\boldsymbol{\theta}}(s',a')\,\mathrm{d}a'\,\mathrm{d}s'$ is the action-value function defined recursively by Bellman's equation. In our case of interest, $\boldsymbol{\theta} = [\boldsymbol{v} \mid w]$ and $\boldsymbol{\theta}' = [\boldsymbol{v}, w']$. We also need the Policy Gradient Theorem (PGT) [15], which we specialize to $\nabla_w J$:

$$\nabla_w J(\boldsymbol{\theta}) = \frac{1}{1-\gamma}\int_{\mathcal{S}} d^{\boldsymbol{\theta}}_{\rho}(s)\int_{\mathcal{A}} \nabla_w \pi_{\boldsymbol{\theta}}(a|s) Q^{\boldsymbol{\theta}}(s,a)\,\mathrm{d}a\,\mathrm{d}s.$$

Plugging the results of Lemmas A.2 and A.3 into (A.9) we get:

$$\begin{aligned} J(\boldsymbol{\theta}') - J(\boldsymbol{\theta}) \geq & \frac{1}{1-\gamma}\int_{\mathcal{S}} d^{\boldsymbol{\theta}}_{\rho}(s)\int_{\mathcal{A}}\left(\nabla_w \pi_{\boldsymbol{\theta}}(a|s)\Delta w - \frac{\psi \Delta w^2}{2\sigma_w}\right) Q^{\boldsymbol{\theta}}(s,a)\,\mathrm{d}a\,\mathrm{d}s \\ & - \frac{\gamma}{2(1-\gamma)^2} 2\Delta w^2 \left\|Q^{\boldsymbol{\theta}}\right\|_{\infty}, \end{aligned} \tag{A.10}$$

[1]This approximation is not critical since the steps produced by safe algorithms tend to be very small.

where $o(\Delta w^3)$ terms, as stated, are neglected. By applying the PGT:

$$\begin{aligned} J(\boldsymbol{\theta}') - J(\boldsymbol{\theta}) &\geq \Delta w \nabla_w J(\boldsymbol{\theta}) - \frac{1}{1-\gamma} \int_{\mathcal{S}} d_\rho^{\boldsymbol{\theta}}(s) \int_{\mathcal{A}} \frac{\psi \Delta w^2}{2\sigma_w} Q^{\boldsymbol{\theta}}(s,a) \, \mathrm{d}a \, \mathrm{d}s \\ &\quad - \frac{\gamma}{2(1-\gamma)^2} 2\Delta w^2 \left\| Q^{\boldsymbol{\theta}} \right\|_\infty \\ &\geq \Delta w \nabla_w J(\boldsymbol{\theta}) - \frac{\psi \Delta w^2}{2\sigma_w (1-\gamma)} \left| \int_{\mathcal{A}} Q^{\boldsymbol{\theta}}(s,a) \, \mathrm{d}a \right| \\ &\quad - \frac{\gamma}{2(1-\gamma)^2} 2\Delta w^2 \left\| Q^{\boldsymbol{\theta}} \right\|_\infty . \end{aligned}$$

Next, following [4], we upper bound the integral and the infinite norm of $Q^{\boldsymbol{\theta}}$ with $\frac{|\mathcal{A}|R}{1-\gamma}$ and $\frac{R}{1-\gamma}$, respectively, obtaining:

$$J(\boldsymbol{\theta}') - J(\boldsymbol{\theta}) \geq \Delta w \nabla_w J(\boldsymbol{\theta}) - \Delta w^2 \frac{R}{(1-\gamma)^2} \left(\frac{\psi |\mathcal{A}|}{2\sigma_w} + \frac{\gamma}{1-\gamma} \right). \tag{A.11}$$

Now, since $\Delta w = \beta \nabla_w J(\boldsymbol{\theta})$,

$$J(\boldsymbol{\theta}') - J(\boldsymbol{\theta}) \geq \beta \nabla_w J(\boldsymbol{\theta})^2 - \beta^2 \nabla_w J(\boldsymbol{\theta})^2 \frac{R}{(1-\gamma)^2} \left(\frac{\psi |\mathcal{A}|}{2\sigma_w} + \frac{\gamma}{1-\gamma} \right).$$

The proof is completed by renaming $\boldsymbol{v} \leftarrow \boldsymbol{v}^{t+1}$, $w \leftarrow w^t$ and $w' \leftarrow w^{t+1}$.

Statement (4.16): The right-hand side of (4.15) is a degree-2 polynomial in β. By nullifying its first derivative:

$$\nabla_w J(\boldsymbol{v}^{t+1}, w^t)^2 - 2\beta^* d \nabla_w J(\boldsymbol{v}^{t+1}, w^t)^2 = 0,$$

we get $\beta^* = \frac{1}{2d}$. Second derivative is $(-\nabla_w J(\boldsymbol{\theta})^2) < 0$, showing that we found a maximum. In fact, we are taking the vertex of a concave parabola. We obtain $C^w_{\max}$ by plugging β^* into the lower bound. Note that we always ignore the degenerate case $\nabla_w J(\boldsymbol{\theta}) = 0$ since it would mean the policy parameters are already at convergence, hence no update should be performed whatsoever.

Statement (4.17): Given (4.15), it is enough to require:

$$\beta \nabla_w J(\boldsymbol{v}^{t+1}, w^t)^2 - \beta^2 d \nabla_w J(\boldsymbol{v}^{t+1}, w^t)^2 \geq \nu C$$

and solve the inequality for β. Note that there is no real solution for $\nu C > C^w_{\max}$. □

Corollary 4.4. *The $\boldsymbol{v}$-update in (4.12) with optimal step size α^* from Theorem 4.2 guarantees:*

$$J(\boldsymbol{v}^{t+1}, w^t) - J(\boldsymbol{v}^t, w^t) \geq \frac{\left\| \nabla_{\boldsymbol{v}} J(\boldsymbol{v}^t, w^t) \right\|_2^2}{4mc} := \mathcal{L}(\boldsymbol{v}^t, w^t),$$

Proof. The corollary follows from Statement (4.9) in Theorem 4.2 and the following norm inequality, true for any m-dimensional vector $\mathbf{x}$:

$$\|\mathbf{x}\|_\infty^2 = \left(\max_i \{|x_i|\} \right)^2 = \frac{1}{m} \sum_{j=1}^m \left(\max_i \{|x_i|\} \right)^2 \geq \frac{1}{m} \sum_{j=1}^m x_j^2 = \frac{1}{m} \|\mathbf{x}\|_2^2 .$$

□

Theorem 4.5. *The w-update in (4.18) guarantees, for any (possibly negative) step size β:*

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^{t}) \geq \beta \nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^{t}) \nabla_w J(\boldsymbol{v}^{t+1}, w^{t}) - \beta^2 d \nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^{t})^2.$$

Guaranteed improvement is maximised by using a step size $\tilde{\beta} = \frac{\nabla_w J(\boldsymbol{v}^{t+1}, w^{t})}{2d \nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^{t})}$, yielding:

$$J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^{t}) \geq C^{w}_{max}.$$

Moreover, for any $\nu C \leq C^{w}_{max}$, the following constraints on the step-size β:

$$|\beta - \tilde{\beta}| \leq \lambda_w |\tilde{\beta}| \tag{4.19}$$

are enough to guarantee $J(\boldsymbol{v}^{t+1}, w^{t+1}) - J(\boldsymbol{v}^{t+1}, w^{t}) \geq \nu C$.

Proof. Rename $\beta \leftarrow \frac{\nabla_w \mathcal{L}(\boldsymbol{v}^{t+1}, w^{t})}{\nabla_w J(\boldsymbol{v}^{t+1}, w^{t})} \beta$ in all the statements to obtain Theorem 4.3. More intuitively, it suffices to consider steps in the direction of $\nabla_w \mathcal{L}$ instead of $\nabla_w J$. Each step in $\nabla_w \mathcal{L}$ has a corresponding (possibly negative) step in $\nabla_w J$, for which Theorem 4.3 applies. Geometrically, we are performing a projection between one-dimensional vector spaces (note that $\nabla_w J$ and $\nabla_w \mathcal{L}$ are scalars). The absolute value in (4.19) is necessary because $\tilde{\beta}$ may be negative. Again, we neglect the zero-gradient case since it coincides with convergence. □

Appendix B

Estimating $\nabla_w \mathcal{L}(\boldsymbol{v}, w)$

In this section we will see how to estimate $\nabla_w \mathcal{L}(\boldsymbol{v}, w)$ from trajectories. To simplify a bit the calculations, we will derive $\nabla_\sigma \mathcal{L}(\boldsymbol{v}, \sigma)$ so that $\nabla_w \mathcal{L}(\boldsymbol{v}, w)$ can be calculated by chain rule:

$$\nabla_w \mathcal{L}(\boldsymbol{v}, w) = e^w \nabla_\sigma \mathcal{L}(\boldsymbol{v}, e^w) \tag{B.1}$$

From definition: $\mathcal{L}(\boldsymbol{\theta}) := \frac{\|\nabla_{\boldsymbol{v}} J(\boldsymbol{\theta})\|_2^2}{4mc} = \frac{1}{2m}\alpha^* \|\nabla_{\boldsymbol{v}} J(\boldsymbol{\theta})\|_2^2$. where α^* is the optimal step size from [4]:

$$\begin{aligned}
\alpha^* &= \frac{c_1 \sigma^3}{m(c_2\sigma + c_3)}, \\
c_1 &= (1-\gamma)^3 \sqrt{2\pi}, \\
c_2 &= \gamma \sqrt{2\pi} R M_\phi^2, \\
c_3 &= 2(1-\gamma)|\mathcal{A}| R M_\phi^2.
\end{aligned}$$

The improvement direction of σ w.r.t. this surrogate objective $\mathcal{L}(\boldsymbol{\theta})$ is:

$$\nabla_\sigma \left(\frac{1}{2m} \alpha^* \|\nabla_{\boldsymbol{v}} J(\boldsymbol{\theta})\|_2^2 \right) = \frac{1}{2m} \nabla_\sigma \alpha^* \|\nabla_{\boldsymbol{v}} J(\boldsymbol{\theta})\|_2^2 + \frac{1}{2m} \alpha^* \nabla_\sigma \|\nabla_{\boldsymbol{v}} J(\boldsymbol{\theta})\|_2^2 .$$

Where we can further derive the two terms:

$$\nabla_\sigma \alpha^* = \frac{2c_1c_2\sigma + 3c_1c_3}{m(c_2\sigma + c_3)^2}\sigma^2 \quad ; \quad \nabla_\sigma \|\nabla_{\boldsymbol{v}} J(\boldsymbol{\theta})\|_2^2 = 2\langle \nabla_{\boldsymbol{v}} J(\boldsymbol{\theta}), \nabla_\sigma \nabla_{\boldsymbol{v}} J(\boldsymbol{\theta}) \rangle.$$

Where $\langle \cdot \rangle$ here denotes scalar product between vectors. Recalling that:

$$\begin{aligned}
\nabla_\sigma \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau) &= \sum_{t=0}^{H} \nabla_\sigma \nabla_{\boldsymbol{v}} \log \pi_{\boldsymbol{\theta}}(a \mid s) = \sum_{t=0}^{H} \nabla_\sigma \frac{a - \mu_{\boldsymbol{\theta}}(s)}{\sigma_{\boldsymbol{\theta}}^2} = -\frac{2}{\sigma} \sum_{t=0}^{H} \frac{a - \mu_{\boldsymbol{\theta}}}{\sigma_{\boldsymbol{\theta}}^2} = \\
&= -\frac{2}{\sigma} \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau),
\end{aligned}$$

we can develop the mixed derivative $\nabla_\sigma \nabla_{\boldsymbol{v}} J(\boldsymbol{\theta})$ using the log-trick:

$$\begin{aligned}
\nabla_\sigma \nabla_{\boldsymbol{v}} J(\boldsymbol{\theta}) &= \nabla_\sigma \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau) R(\tau)\right] = \\
&= \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}} \left[\nabla_\sigma \log p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau) R(\tau)\right] + \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}} \left[\nabla_\sigma \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau) R(\tau)\right] \\
&= \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}} \left[\nabla_\sigma \log p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau) R(\tau)\right] - \frac{2}{\sigma} \nabla_{\boldsymbol{v}} J(\boldsymbol{\theta}) \\
&= h(\boldsymbol{\theta}) - \frac{2}{\sigma} \nabla_{\boldsymbol{v}} J(\boldsymbol{\theta}),
\end{aligned}$$

where $h(\boldsymbol{\theta}) := \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}} \left[\nabla_\sigma \log p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau) R(\tau)\right]$.

B.1 Estimating $h(\boldsymbol{\theta})$

In this part we will derive an estimator for $h(\boldsymbol{\theta})$.

Theorem B.1. *Assuming Gaussian policies, an unbiased estimator for*

$$h(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}} \left[\nabla_\sigma \log p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau) R(\tau)\right] \tag{B.2}$$

is:

$$\hat{h}(\boldsymbol{\theta}) = \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}(\tau_{0:H}) \gamma^t r_t \left(\sum_{k'=0}^{t} \nabla_\sigma \log \pi_{k'}\right) \left(\sum_{h'=0}^{t} \nabla_{\boldsymbol{v}} \log \pi_{h'}\right) d\tau \tag{B.3}$$

Proof. Let's denote $\pi_k = \pi_{\boldsymbol{\theta}}(a_k \mid s_k)$. Then, by splitting the trajectory and using the same observation as in GPOMDP estimator in [1], the estimator $h(\boldsymbol{\theta})$ can be splitted in the sum of four components:

$$\begin{aligned}
h(\boldsymbol{\theta}) &= \int_T p_{\boldsymbol{\theta}}(\tau_{0:H}) \nabla_\sigma \log p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau) R(\tau)\, \mathrm{d}\tau = \\
&= \int_T p_{\boldsymbol{\theta}}(\tau_{0:H}) \left(\sum_{t=0}^{H} \gamma^t r_t\right) \left(\sum_{k=0}^{H} \nabla_\sigma \log \pi_k\right) \left(\sum_{h=0}^{H} \nabla_{\boldsymbol{v}} \log \pi_h\right) \mathrm{d}\tau = \\
&= \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}(\tau_{0:H}) \gamma^t r_t \left(\sum_{k'=0}^{t} \nabla_\sigma \log \pi_{k'}\right) \left(\sum_{h'=0}^{t} \nabla_{\boldsymbol{v}} \log \pi_{h'}\right) \mathrm{d}\tau + \qquad \text{(B.4)} \\
&+ \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}(\tau_{0:H}) \gamma^t r_t \left(\sum_{k'=0}^{t} \nabla_\sigma \log \pi_{k'}\right) \left(\sum_{h''=t+1}^{H} \nabla_{\boldsymbol{v}} \log \pi_{h''}\right) \mathrm{d}\tau + \qquad \text{(B.5)} \\
&+ \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}(\tau_{0:H}) \gamma^t r_t \left(\sum_{k''=t+1}^{H} \nabla_\sigma \log \pi_{k''}\right) \left(\sum_{h'=0}^{t} \nabla_{\boldsymbol{v}} \log \pi_{h'}\right) \mathrm{d}\tau + \qquad \text{(B.6)} \\
&+ \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}(\tau_{0:H}) \gamma^t r_t \left(\sum_{k''=t+1}^{H} \nabla_\sigma \log \pi_{k''}\right) \left(\sum_{h''=t+1}^{H} \nabla_{\boldsymbol{v}} \log \pi_{h''}\right) \mathrm{d}\tau. \qquad \text{(B.7)}
\end{aligned}$$

Next, we will show that (B.5), (B.6) and (B.7) will both evaluate to 0:

$$\begin{aligned}
(B.5) &= \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:H}\right) \gamma^t r_t \left(\sum_{k'=0}^{t} \nabla_\sigma \log \pi_{k'}\right)\left(\sum_{h''=t+1}^{H} \nabla_{\boldsymbol{v}} \log \pi_{h''}\right) \mathrm{d}\tau = \\
&= \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t \left(\sum_{k'=0}^{t} \nabla_\sigma \log \pi_{k'}\right) \mathrm{d}\tau \int_T p_{\boldsymbol{\theta}}\left(\tau_{t+1:H}\right)\left(\sum_{h''=t+1}^{H} \nabla_{\boldsymbol{v}} \log \pi_{h''}\right) \mathrm{d}\tau = \\
&= \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t \left(\sum_{k'=0}^{t} \nabla_\sigma \log \pi_{k'}\right) \mathrm{d}\tau \int_T p_{\boldsymbol{\theta}}\left(\tau_{t+1:H}\right) \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}\left(\tau_{t+1:H}\right) \mathrm{d}\tau = \\
&= \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t \left(\sum_{k'=0}^{t} \nabla_\sigma \log \pi_{k'}\right) \mathrm{d}\tau \int_T \nabla_{\boldsymbol{v}} p_{\boldsymbol{\theta}}\left(\tau_{t+1:H}\right) \mathrm{d}\tau = \\
&= \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t \left(\sum_{k'=0}^{t} \nabla_\sigma \log \pi_{k'}\right) \mathrm{d}\tau \nabla_{\boldsymbol{v}} \int_T p_{\boldsymbol{\theta}}\left(\tau_{t+1:H}\right) \mathrm{d}\tau = \\
&= 0
\end{aligned}$$

Using the same approach of (B.5) we can say that (B.6) = 0

$$\begin{aligned}
(B.7) &= \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:H}\right) \gamma^t r_t \left(\sum_{k''=t+1}^{H} \nabla_\sigma \log \pi_{k''}\right)\left(\sum_{h''=t+1}^{H} \nabla_{\boldsymbol{v}} \log \pi_{h''}\right) \mathrm{d}\tau = \\
&= \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t \, \mathrm{d}\tau \int_T p_{\boldsymbol{\theta}}\left(\tau_{t+1:H}\right) \nabla_\sigma \log p_{\boldsymbol{\theta}}\left(\tau_{t+1:H}\right) \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}\left(\tau_{t+1:H}\right) \mathrm{d}\tau = \\
&= \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t \, \mathrm{d}\tau \int_T \left(\nabla_\sigma \nabla_{\boldsymbol{v}} p_{\boldsymbol{\theta}}(\tau_{t+1:H}) - \nabla_\sigma \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau_{t+1:H})\right) \mathrm{d}\tau = \\
&= \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t \, \mathrm{d}\tau \int_T \nabla_\sigma \nabla_{\boldsymbol{v}} p_{\boldsymbol{\theta}}(\tau_{t+1:H}) \, \mathrm{d}\tau - \\
&\qquad - \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t \, \mathrm{d}\tau \int_T \nabla_\sigma \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau_{t+1:H}) \, \mathrm{d}\tau = \\
&= \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t d\tau \nabla_\sigma \nabla_{\boldsymbol{v}} \int_T p_{\boldsymbol{\theta}}(\tau_{t+1:H}) \, \mathrm{d}\tau - \\
&\qquad - \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t \, \mathrm{d}\tau \int_T -\frac{2}{\sigma} p_{\boldsymbol{\theta}}(\tau_{t+1:H}) \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau_{t+1:H}) \, \mathrm{d}\tau = \\
&= \frac{2}{\sigma} \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t \, \mathrm{d}\tau \int_T p_{\boldsymbol{\theta}}(\tau) \nabla_{\boldsymbol{v}} \log p_{\boldsymbol{\theta}}(\tau_{t+1:H}) \, \mathrm{d}\tau = \\
&= \frac{2}{\sigma} \sum_{t=0}^{H} \int_T p_{\boldsymbol{\theta}}\left(\tau_{0:t}\right) \gamma^t r_t \, \mathrm{d}\tau \int_T \nabla_{\boldsymbol{v}} p_{\boldsymbol{\theta}}(\tau_{t+1:H}) \, \mathrm{d}\tau = 0.
\end{aligned}$$

□

B.2 Baseline for $\hat{h}(\boldsymbol{\theta})$

Let $\hat{h_t}(\boldsymbol{\theta}) = \mathbb{E}\left[\underbrace{\left(\sum_{k=0}^{t} \nabla_\theta \log \pi_k\right)}_{G_t} \underbrace{\left(\sum_{k=0}^{t} \nabla_\sigma \log \pi_k\right)}_{H_t} \left(\underbrace{\gamma^t r_t}_{F_t} - b_t\right)\right]$ where b_t is a generic baseline that is independent w.r.t. the actions a_k.

Any baseline $b_t = \tilde{b_t}\left(\frac{G_t+H_t}{G_t H_t}\right)$ will keep the estimator unbiased for any value of $\tilde{b_t}$:

$$\begin{aligned} E\left[G_t H_t \left(F_t - b_t\right)\right] &= E\left[G_t H_t F_t\right] - E\left[G_t H_t b_t\right] = \\ &= E\left[G_t H_t F_t\right] - E\left[G_t H_t \tilde{b_t}\left(\frac{G_t + H_t}{G_t H_t}\right)\right] = \\ &= E\left[G_t H_t F_t\right] - \tilde{b_t} E\left[G_t\right] - \tilde{b_t}\left[H_t\right] = \\ &= E\left[G_t H_t F_t\right]. \end{aligned}$$

We choose $\tilde{b_t}$ to minimize the variance of $\hat{h_t}$:

$$\begin{aligned} Var[\hat{h_t}] &= E\left[\hat{h_t}^2\right] - E\left[\hat{h_t}\right]^2 = \\ &= E\left[G_t^2 H_t^2\left(F_t^2 - 2F_t\tilde{b_t}\frac{G_t + H_t}{G_t H_t} + \tilde{b_t}^2\frac{(G_t + H_t)^2}{G_t^2 H_t^2}\right)\right] - E\left[G_t H_t F_t\right]^2 = \\ &= E\left[G_t^2 H_t^2 F_t^2\right] - 2\tilde{b_t} E\left[G_t H_t F_t\left(G_t + H_t\right)\right] + \tilde{b_t}^2 E\left[\left(G_t + H_t\right)^2\right] - E\left[G_t H_t F_t\right]^2 \end{aligned}$$

$$b_t^* = \arg\min_{\tilde{b_t}} Var\left[\hat{h_t}\right] = \frac{E\left[G_t H_t F_t\left(G_t + H_t\right)\right]}{E\left[\left(G_t + H_t\right)^2\right]}.$$

Hence the estimator has minimum variance with baseline:

$$b_t = b_t^* \frac{G_t + H_t}{G_t H_t} = \frac{\langle G_t H_t F_t\left(G_t + H_t\right)\rangle}{\left\langle\left(G_t + H_t\right)^2\right\rangle}\frac{G_t + H_t}{G_t H_t}$$